DYNAMIC RESOURCE MANAGEMENT IN RSVP-CONTROLLED UNICAST NETWORKS

Venkatesan Iyengar Prasanna, B.E.

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2001

APPROVED:

Armin R. Mikler, Major Professor
Azzedine Boukerche, Committee Member
Paul Tarau, Committee Member
Tom Jacob, Committee Member
Krishna Kavi, Chair of the Department of Computer Science
C. Neal Tate, Dean of the Robert B. Toulouse School of Graduate Studies

Iyengar Prasanna, Venkatesan, Dynamic Resource Management in RSVP-Controlled Unicast Networks. Master of Science (Computer Science), December 2001, 94 pp., 25 figures, 22 titles.

Resources are said to be fragmented in the network when they are available in non-contiguous blocks, and calls are dropped as they may not find sufficient resources. Hence, available resources may remain unutilized. In this thesis, the effect of resource fragmentation (RF) on RSVP-controlled networks was studied and new algorithms were proposed to reduce the effect of RF. In order to minimize the effect of RF, resources in the network are dynamically redistributed on different paths to make them available in contiguous blocks. Extra protocol messages are introduced to facilitate resource redistribution in the network. The Dynamic Resource Redistribution (DRR) algorithm when used in conjunction with RSVP, not only increased the number of calls accommodated into the network but also increased the overall resource utilization of the network. Issues such as how many resources need to be redistributed and of which call(s), and how these choices affect the redistribution process were investigated. Further, various simulation experiments were conducted to study the performance of the DRR algorithm on different network topologies with varying traffic characteristics.

ACKNOWLEDGEMENTS

I would like to express my profound gratitude and appreciation to my research advisor, Dr. Armin Mikler, for his patient, encouraging and enthusiastic support and guidance throughout this endeavor. The discussions and arguments I had with Dr Mikler helped me gain more insight into my area of research and this would not have been possible without his unique style of looking at any questions from students with an open mind and infinite patience.

I would also like to thank my committee members Dr. Azzedine Boukerche, Dr. Paul Tarau and Dr. Tom Jacob for sparing their time from their busy schedules to be part of my thesis defence committee and giving valuable suggestions.

A special note of appreciation is extended to Sandhya for her instrumental help in drafting this thesis. I am extremely thankful to her for meticulously proofreading my thesis and giving numerous suggestions.

I am grateful to Cliff and Sandeep for sharing their simulation engine and graph generator respectively. A special note of thanks is extended to John for helping me overcome various Linux and C++ related problems. I am thankful to Glyco, Kaizar and Subhashini for reviewing the structural representation of the algorithms presented in this thesis. I would like to thank the entire NRL group for various suggestions made on network related issues at appropriate times. I would also like to thank Dr.Tate for helping me overcome various C++ implementation problems, and Dr.Böhme and Hong for their help on Latex.

In particular, I am extremely grateful my roommates Vivek, Laxmi and Harita for cooking food, substituting my cooking turns and co-operating with me in every possible way thereby helping me spend more time in my research.

Last but not the least, I would like to thank my parents for their persistent help, guidance and inspiration without which I would not be in USA pursuing a Master's degree today. I would like to thank my sister and brother-in-law for their love and moral support. I would also like to thank all my relatives, friends and well wishers for their unstinted support and co-operation at various points of time.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

In the past ten years, we have witnessed an unprecedented growth of the Internet. With the advent of the World Wide Web (WWW), the use of the Internet is no longer restricted to the academic domain only. Many corporate houses have already started using the Internet for commercial purposes. The available network bandwidth has also increased over the years. We now have Broadband Integrated Services Digital Network (B-ISDN), Asynchronous Transfer Mode and Gigabit Ethernet technologies at our disposal. In fact, merely increasing the bandwidth does not solve the problem of timeliness of delivery or variation in delay (jitter) because as more bandwidth is introduced, more traffic is generated to utilize it.

The growth of the Internet along with the arrival of the WWW has produced the following effects: The number of users in the network has increased to millions. Consequently there is more traffic being induced into the network. If the rate of incoming traffic is higher than that of outgoing traffic in the routers and switches, they discard the packets. This is due to the fact that the routers and switches

have finite processing power and buffer space. If packet discards occur frequently, then the network is said to be congested. Thus, increased growth of the network without efficient management of resources may lead to congestion. The second effect is that new types of traffic such as audio traffic and video traffic are introduced in the network. Applications like audio and video playback are jitter sensitive whereas Internet Telephony applications are delay sensitive. The traditional Internet Protocol (IP) does not guarantee reliable delivery of packets to the receiver.

As mentioned earlier, network elements like routers and switches will drop packets if they cannot accommodate the incoming traffic. Although reliable delivery of packets can be provided by higher layer transport protocols like Transmission Control Protocol (TCP) at the end hosts, timely delivery or jitter free traffic cannot be guaranteed by any of the higher layer protocols [19]. This is due to the fact that jitter and delay are introduced in the network because of congestion in certain areas of the network. End-hosts generally have no control over these areas. However, TCP can adjust itself to congestion by reducing its window size, but this is done only after congestion is detected in the network and not before congestion occurs.

Internet traffic can be broadly classified as follows: Real-time traffic and non real-time traffic. Real-time traffic, such as video conferencing, real-time multimedia, etc requires a short transmission delay and very little or no jitter. Video applications

are jitter sensitive since the sending rate and receiving rate must be synchronized. The jitter can be neutralized by buffering the packets for a certain period of time and then releasing the packets at a constant rate to the receiver application [15]. The synchronization provided by the buffer is dependent on the expected maximum and minimum end-to-end delay. For example, consider Figure 1.1 where packet P3 is lost in the network. If the maximum end-to-end delay (MAX_D) is 9 ms and the minimum end-to-end delay (MIN_D) is 3 ms, the incoming packets can be delayed up to the difference between MAX_D and MIN_D, which is 6 ms in this case. When the packets P1, P2 and P4 reach the receiver, they are buffered for 6 ms. When P3 does not reach the receiver within 6 ms, the receiver does not wait for it any longer. However, if the sender retransmits P3 within 6 ms, it will not be discarded by the receiver.

Non real-time traffic can tolerate higher delays as compared to real-time traffic. Reliability and throughput are of concern for these types of traffic. Reliability can be achieved by using timer and acknowledgement mechanisms in the higher layers at the end hosts. When a packet is lost in the network, the receiver does not acknowledge the packet, forcing the sender to timeout and retransmit the packet. Conventional Internet applications like email, ssh and HTTP fall into this category of traffic.

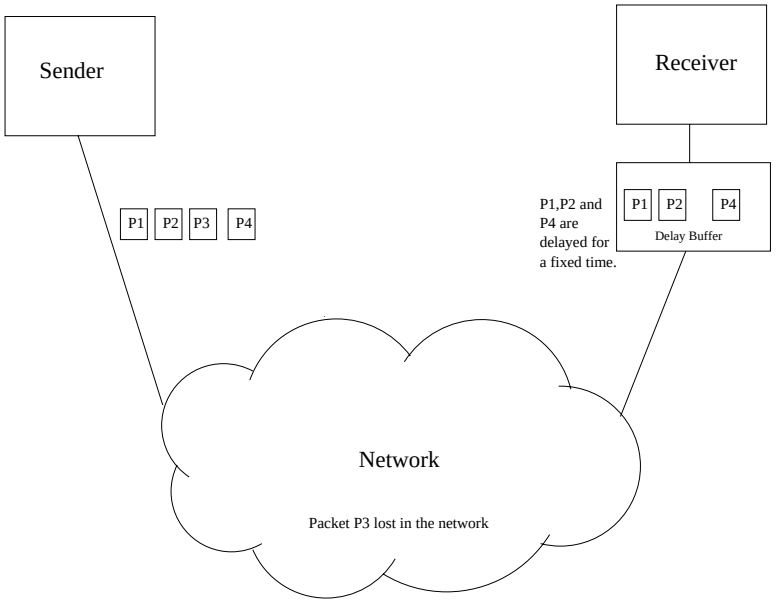


Figure 1.1: Real-time communication

IP services packets on a best-effort basis, that is, all packets are treated alike. Best-effort service does not prevent any traffic from entering the network. When the network load is light, packet discards in the network are rare. But as the load increases, the number of packets dropped also increases, thereby affecting all the traffic passing through the congested area in the network [7]. This type of service cannot ensure timely delivery of a packet to a destination. Therefore, when real-time traffic is sent across the Internet, the performance of a real-time application at the end host will be negatively affected. One way to overcome the above problem is to treat different packets differently. Quality of Service (QoS) facilitates prioritization

of various traffic to ensure that high priority applications always get the network resources to send their traffic successfully. QoS is not just about prioritizing traffic but also about the efficient management of resources by means of admission control and congestion management mechanisms.

1.2 Definition of QoS

The objective of QoS is to introduce unfairness in the network. In the best-effort model, all packets are treated equally (fairly). When all packets in the network are treated alike, many low priority packets may clog the network and force the routers and switches to drop high priority packets. Thus, fairness may result in negatively affecting high priority traffic. In other words, QoS is about giving different levels of preferential treatment to different types of traffic depending on the policy of the network service provider. As defined in [7], *" QoS can be interpreted as method to provide preferential treatment to some arbitrary amount of network traffic, as opposed to all traffic being treated as "best effort," and in providing such preferential treatment, attempting to increase the quality level of one or more of these basic metrics for this particular category of traffic."*

QoS can also be defined as a quantified approach to provide preferential treatment to different types of traffic based on a desired traffic parameter. The key traffic parameters are *delay*, *jitter*, *bandwidth* and *packet loss rate*.

Delay is defined as the time taken by a packet to traverse the network from sender to receiver. This delay includes transmission delay, propagation delay and queuing delay (time spent waiting in the queues of network elements).

Jitter, as explained earlier, is defined as the variation in delay experienced by the packets. When packets are transmitted using IP, different packets take different routes. Some traverse the congested area of the network experiencing delay whereas other packets travel through the non-congested areas of the network. This results in the non-sequential arrival of packets at the destination. This variation in delay is referred to as jitter. Applications such as playback video are jitter sensitive as the sending and receiving rates need to be synchronized.

Bandwidth is the maximum amount of data that can be sent between two end points of a link during a given period of time. Transmission delay depends on the bandwidth of a link. The larger the bandwidth, the more data that can be sent over a link.

Packet loss rate is defined as the number of packets lost per unit time. While requesting QoS, a user or an application can specify the maximum packet loss rate that

is deemed acceptable. If the service provider is not able to guarantee the requested packet loss rate, it can ask the user to retry after some time.

QoS based on the above traffic parameters can be provided by selecting an appropriate next hop in the router, by changing the the packet-discard algorithms in the queue of routers [7] or by reserving resources in the network based on the service criteria published by the service provider. The QoS requesting application can advertise its traffic behavior to enable the network to allocate resources for it. It is the responsibility of the network to police the traffic to ensure that it conforms to its traffic profile. If an application violates its advertised specification, its packets may be serviced on a best-effort basis or dropped. QoS brings a certain degree of discipline to the manner in which the sender sends its traffic and the way it obtains service from the network.

QoS provides control over the resources of the network. For example, we can limit the bandwidth consumed by telnet applications and can dedicate a higher percentage of bandwidth to Hyper Text Transfer Protocol (HTTP) [5]. QoS also ensures that low priority applications also get a fair share of bandwidth without affecting mission critical applications. QoS does not create bandwidth but manages the resources effectively to meet the wide-ranging application requirements [17].

1.3 QoS Mechanisms

To provide consistent service to users, the network needs to classify packets, give preferential treatment based on a traffic parameter and police the flows to ensure that they conform to their traffic profile. All these functions are not performed by a single module at the same time but by different modules at different points in time. For example, the admission control algorithm is applied when a new connection request arrives in the network. The congestion management algorithm is enforced when the network is about to become congested. Every time a packet arrives at a router, the service to be given to that packet is identified by looking at its header. Packet classifiers partition the network into different levels of traffic. A three-bit Type of Service (TOS) field in the IP packet header can be used to classify different levels of service. After a packet is classified, the appropriate packet discard algorithm, congestion management functions and bandwidth allocation algorithm can be applied for it at the appropriate time. Each TOS field value identifies and associates a unique service applied to a packet in each node along the path traversed by the packet [19]. Each router orders the packets based on the TOS field.

Congestion management refers to actions that are taken after congestion occurs and also to actions that are taken to prevent congestion. Congestion occurs in the

network when the packet arrival rate exceeds the packet departure rate of the network resulting in discarding of packets. TCP uses a slow start algorithm as a remedial measure after congestion occurs. Congestion avoidance techniques monitor the network traffic regularly to determine the likelihood of congestion in the future. Congestion avoidance algorithms such as Random Early Detection (RED) [9] avoid congestion by randomly dropping packets from the queue in order to avoid global synchronization. The word *global* here refers to all the flows experiencing congestion at the same time. When congestion occurs due to many active flows in the network, the packet discard experienced by all the flows forces each one of them to back off (apply the slow start algorithm) and then begin retransmitting at the same time. This trend leads to an oscillation between periods of congestion and no transmission of traffic. In fact, this phenomenon does not help the network in reducing the congestion. RED avoids global synchronization by discarding the packet of a single flow rather than of multiple flows. RED monitors the mean queue depth and as it starts approaching the predefined maximum limit, it randomly selects a packet and drops it, thus signaling the corresponding sender to slow down its transmission rate.

Admission control algorithms determine whether a new QoS request can be accommodated in the network without affecting the existing QoS commitments. Admission control modules limit the amount of traffic entering the network, thereby controlling

the degree of congestion introduced in the network [8]. Admission control can be done on a per flow or per packet basis. In the first case, the admission control module is invoked only when a flow tries to reserve resources during the signaling phase. Once resources are allocated, this module is never called again. In the latter, the admission control module is invoked for every packet a node receives, i.e., when a packet reaches a node, a decision is made whether this packet can be forwarded or not. If the admission control does not succeed, the packet is dropped.

Policies [18] are the set of rules and information databases that help the network to translate its administrative constraints to the degree of preferential treatment given to a packet. When a packet reaches a node, the policy to be applied is decided on by looking at its header. Policies determine the resource access rights for users, applications or hosts and also determine under what conditions they are applicable to them.

In other words, policies are used to check whether the users have the administrative permission to access the resources or not. For example, a node N may have a policy where all email traffic coming from link X will be sent over the 10Mbps outgoing link whereas the same type of traffic coming from link Y will be forwarded to the next hop through 1 Mbps link (see Figure 1.2). In particular, policies help network administrators to maintain the network resources in an orderly fashion and

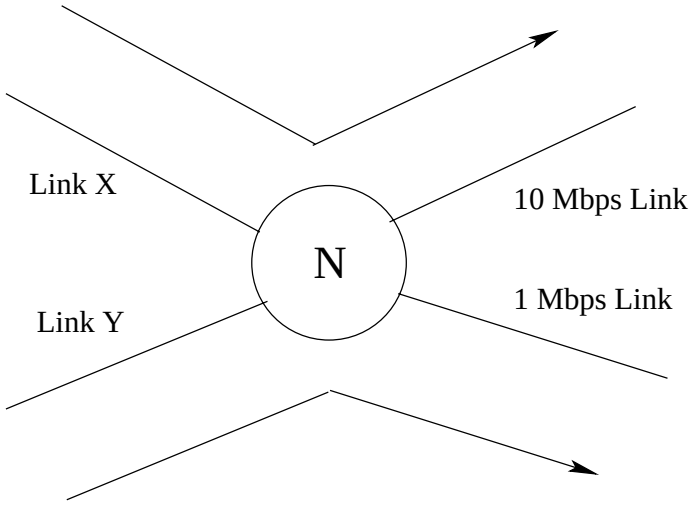


Figure 1.2: Policy : An Example

also regulate access to the resources.

The policy framework (see Figure 1.3) contains three components [18]. Policy Enforcement Point (PEP) is the component responsible for enforcement of policies on packets. Policies can be enforced in each router of a network or they can be enforced on the fringes of a network. Local policies are valid only within the local network. The Internet constitutes many autonomous systems with wide ranging policies. The policies of two different networks might complement each other. For example, one network might give low priority to HTTP traffic whereas some other network might

give a higher priority to the same type of traffic. If a packet traverses these two networks, it will not be treated consistently. To overcome this problem, each autonomous system has a bandwidth broker, and bandwidth brokers of different autonomous systems negotiate with each other before forwarding packets. The autonomous systems may have Service Level Agreements (SLA) between each other that specify how traffic from different autonomous systems needs to be treated. Policy Decision Point (PDP) is the component that determines what policies are applicable to various packets. The policy applied depends on the header of the packet. For example, when the TOS field in the IP header is used to differentiate between various classes of traffic or service, specific policies to be applied for all possible values of the TOS field should be defined by the service provider. When a packet reaches a node, the appropriate policy will be selected in accordance with its TOS field value.

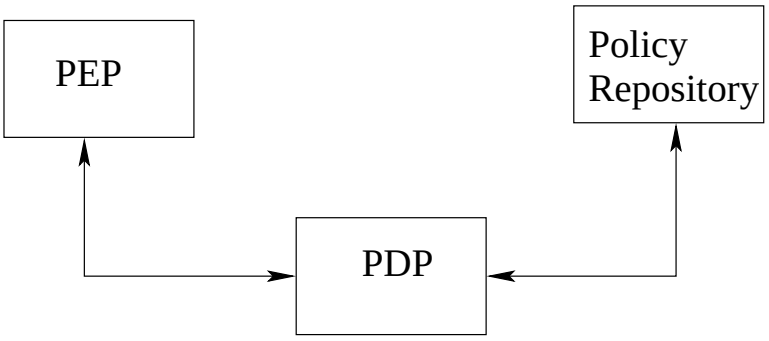


Figure 1.3: Policy Framework [18]

A Policy Repository is the place where policies are stored. The representation of policies should be the same across all networks to give consistent service to applications. The policy schema has three types of entries: policies, conditions and actions [12]. Each schema contains an *"if condition, then action"* policy. For instance, *"if the destination address is 129.120.34.56, then insert this packet in the highest priority output queue"* is an example of a policy. Conditions determine the classes of traffic whereas actions determine the rules that identify the specific service to be given to that packet. A policy management tool can be used to store policies in the policy repository. The policy management tool can also be used for other purposes like testing consistencies of policy, etc,.

The two most popular QoS architectures are the Differentiated Service Architecture (DiffServ) and the Integrated Service Architecture (IntServ). DiffServ is intended to give scalable service discrimination in the network without the need for maintenance of a flow state and signaling in the node [8]. Different services are identified by looking at appropriate bits in the IP header fields and these bits determine how packets are treated in the network. IntServ on the other hand, assumes that control of resources is required to deliver QoS [8]. In IntServ, an application requests a specific level of service before it starts transmitting data. The application advertises its traffic specifications to the network and requests a specific QoS to satisfy its bandwidth

and delay requirements [5]. The application starts transmitting data only after its request has been accepted by the network. The network executes admission control to determine whether this specific QoS request can be granted without violating the QoS commitments given to other flows in the network. The Resource Reservation Protocol (RSVP) is an IntServ signaling protocol that is used to allocate and release resources in the network. The possible consequence of reserving resources in the network is that it may lead to resource fragmentation if resources are not managed properly. Resource fragmentation (RF) occurs when resources are available in the network but cannot be utilized because they are not available in contiguous blocks along the path requested. This situation is similar to disk and memory fragmentation, for which some algorithms are applied to maximize the amount of contiguous resource (memory or disk) space.

For example, assume that the resources are requested in terms of bandwidth. Also assume that node A wants to send data to node H, and requests a bandwidth of 7 Mbps on each of the intermediate links on its path to node H (Figure 1.4). This means that there needs to be 7 Mbps of bandwidth available on each link of the path between node A and node H. As seen in the Figure 1.4, there is 7 Mbps of available bandwidth from node A to node B. However, at node B we see that there are 3 different routes that can be taken, none of which has at least 7 Mbps to be allocated

to this resource request. The total available bandwidth of 11 Mbps (4 Mbps on link BD + 3 Mbps on link BC + 4 Mbps on link BE) has been fragmented and cannot be used, leading to the reservation request failing and the connection being dropped.

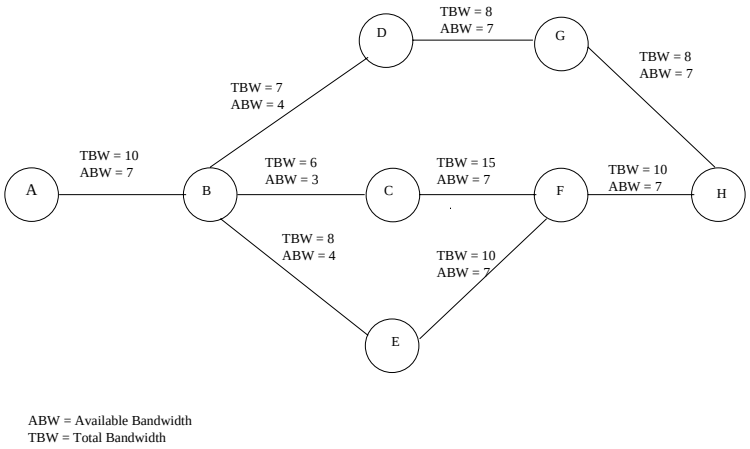


Figure 1.4: Fragmentation

Resource space fragmentation cannot be eliminated completely in spite of its undesirability. This is due to the random arrival and departure of connections and non-uniformity of resource requests that are independently catered to, leading to allocation of resources as and when available, usually on a first-come, first-served basis regardless of RF consequences. Moreover, signaling mechanisms and routing protocols are independent of each other, which may to lead to RF. Popular routing protocols like Routing Information Protocol (RIP) use number of hops as the metric. This means that the path with the least number of hops to the destination is selected

as the route. This may result in many connections trying to use the limited resources along the links of the shortest path, and connections being dropped as resources may not be available along the shortest path. This is a direct consequence of resource fragmentation because discrete chunks of resources on the selected path are not available, but there may be plentiful resources along other sub-optimal paths to the destination.

Combining routing protocols and resource reservation signaling mechanisms to avoid RF is not a viable solution because routing is a network layer function whereas signaling is a transport layer function. In fact signaling is done between end hosts whereas routing is done in the network elements like routers and switches, which makes it impossible to integrate the functionality of both.

Dynamic redistribution of resources in the network may be a feasible solution to minimize the degree of RF. Thus, in figure 1.4, when the resource request for 7 Mbps fails at node B, resources are attempted to be redistributed from one link to another, such that at least one link can have the full 7 Mbps to satisfy the current resource request. In this case, if 3 Mbps currently used can be transferred from link BE or from link BD, to any of one or more of the other links, that link (BE or BD respectively) will have the full 7 Mbps required to satisfy the current request. For example, if link BE transfers a total of 3 Mbps to link BC, then link BC is now full, and link BE has an available bandwidth of 7 Mbps to fulfill the request (The transfer of 3 Mbps from

link BE may be in parts of 1 Mbps and 2 Mbps respectively to links BC and BD, or in any other permutation).

In this thesis, an algorithm has been devised for the dynamic redistribution of resources when Resource Reservation Protocol (RSVP) [2] is used as a signaling protocol. The nodes concerned exchange messages to redistribute resources without affecting the other flows present in the network. Only unicast communications have been considered in this work. RSVP allocates and releases resources by sending and receiving protocol specific messages; a few extra messages have been added to the protocol to aid the redistribution of resources.

1.4 Overview

Chapter 2 describes RSVP in detail. The various special messages and the underlying algorithms used for dynamic management of resources are described in Chapter 3. The simulation results of the proposed algorithms are discussed Chapter 4. The summary of this thesis, with some suggestions for future work, is presented in Chapter 5.

CHAPTER 2

RSVP

2.1 Introduction

Many architectures have been designed to incorporate Quality of Service approaches in networks. The two most popular architectures are Differentiated Service Architecture (DiffServ) and Integrated Service Architecture (IntServ). DiffServ attempts to provide QoS in the Internet without storing the flow state information for active flows in the network. Services are constructed by setting up appropriate bits in the Internet Protocol (IP) header fields. These bits are used to determine how a packet is forwarded in the network. Since this architecture does not reserve any resources in the network, it is not suitable for hard real time applications. On the other hand, the fundamental principle of IntServ is that the resources in the network must be regulated in order to provide QoS [3]. In other words, all traffic must be subjected to an admission control mechanism before resources are allocated to them. IntServ is an extension of the basic IP service and it uses the underlying IP infrastructure to provide resource reservations and service guarantees. IntServ offers two types of service: controlled load service (CLS) and guaranteed load service (GLS). These two

services are explained in detail in [20] and [13] respectively. As explained in [20], CLS closely approximates the end-to-end traffic behavior of a best effort service model when the network is not congested. If the network is functioning correctly, applications using CLS may assume that a very high percentage of packets will be delivered successfully to the receiver node and the delay experienced by the majority of the transmitted packets will rarely exceed the minimum delay experienced by any successfully transmitted packet. CLS is intended to support applications that are sensitive to overloaded conditions. It uses admission control mechanisms before accepting a request for its service, and does not accept any flow that may lead to overloading of the network. To accomplish the above, the application requesting the CLS should provide the network with a traffic specification, which is an estimation of traffic it will generate in the network. If the application violates its traffic specification, the corresponding packets may be dropped, or serviced on a best effort basis. Intermediate network elements can employ admission control or any scheduling algorithm to ensure that the admitted packet flows receive the appropriate service.

The fundamental principle that guided the creation of the Internet was that the state information of flows must be stored only in the end systems [6]. GLS of IntServ deviates from this principle and stores per-flow state information in the network in order to allocate resources. As explained in [13], GLS is used for those applications

that require a bandwidth guarantee and delay bounds. This service neither controls the transmission delay that is caused due to the limited capacity of links along the path nor does it control the jitter. GLS approximates the behavior of a dedicated virtual circuit between two nodes. This is accomplished by using a signaling protocol that reserves resources in the network. The resources are reserved in the network on the basis of the traffic profile advertised by the sender node. RSVP is one such signaling protocol that can allocate and release resources, and can also support other control mechanisms required to provide GLS. This chapter describes the messages and concepts of RSVP that are relevant to this thesis.

2.2 Resource Reservation Protocol

RSVP is a signaling protocol that facilitates Internet applications to get Quality of Service from the network. As explained in [2], RSVP is an Internet control protocol similar to Internet Control Management Protocol (ICMP) and Internet Group Management Protocol (IGMP), and cannot be used to send application data. RSVP operates above the IP layer and is designed to work with the routing protocol implemented in the IP layer. Application data is carried by transport protocols, such as UDP, and they interact with RSVP to obtain a QoS guarantee.

2.2.1 The Workings of RSVP

In RSVP, a sender starts sending data only after its resource request is granted by the network. The network reserves resources for a flow based on the traffic specification (TSpec) advertised by its sender. The sender sends its traffic specification using a $PATH$ message. The $PATH$ message establishes a route between the sender and the receiver, where resources may be allocated. A path state characterization is maintained in each node along the path between the sender and the receiver. Each path state in a node is specific to a particular sender's flow that includes at least a unicast address of the previous hop node. This previous hop information is used to route the $RESV$ message in the reverse direction from the receiver to the sender. In RSVP, the receiver is responsible for requesting QoS and this is done not only to efficiently accommodate large multicast groups but also to facilitate dynamic group memberships for multicast, and to accommodate diverse receiver requirements. After the $PATH$ message reaches the destination, the receiver sends a $RESV$ message to the sender. The $RESV$ message allocates resources and maintains a reservation state in each node along the reverse path towards the sender. The *forward path* refers to the direction from the sender to the receiver whereas the *reverse path* refers to the direction from the receiver to the sender. When the $RESV$ message reaches the sender, the sender starts sending its data. Every RSVP resource request consists of

a flow-spec and a filter-spec, which are called flow descriptors. The flow-spec defines the requested QoS whereas filter-spec defines the set of data packets that will receive the QoS service defined in flow-spec. The filter-spec is a tuple represented as (source address, source port).

QoS for a particular data flow is implemented collectively by a set of components called traffic control (Figure 2.1 [1]) in each node. The four collective components that constitutes traffic control are as follows:

1. Admission Control
2. Policy Control
3. Packet Classifier
4. Packet Scheduler

When an RSVP QoS request reaches a node, the admission control and the policy control modules are invoked first. The admission control interprets the QoS request and decides whether the node has sufficient resources to grant the requested QoS. It should also ascertain that granting the current QoS request does not negatively affect the other established flows. The policy control module decides whether the user has the administrative permission to request the QoS.

When the data packets associated with an established resource reservation reach a node, the packet classifier orders the data packets according to the filter-spec and

[1] Traffic control diagram reproduced from RFC 2205 with editor's permission

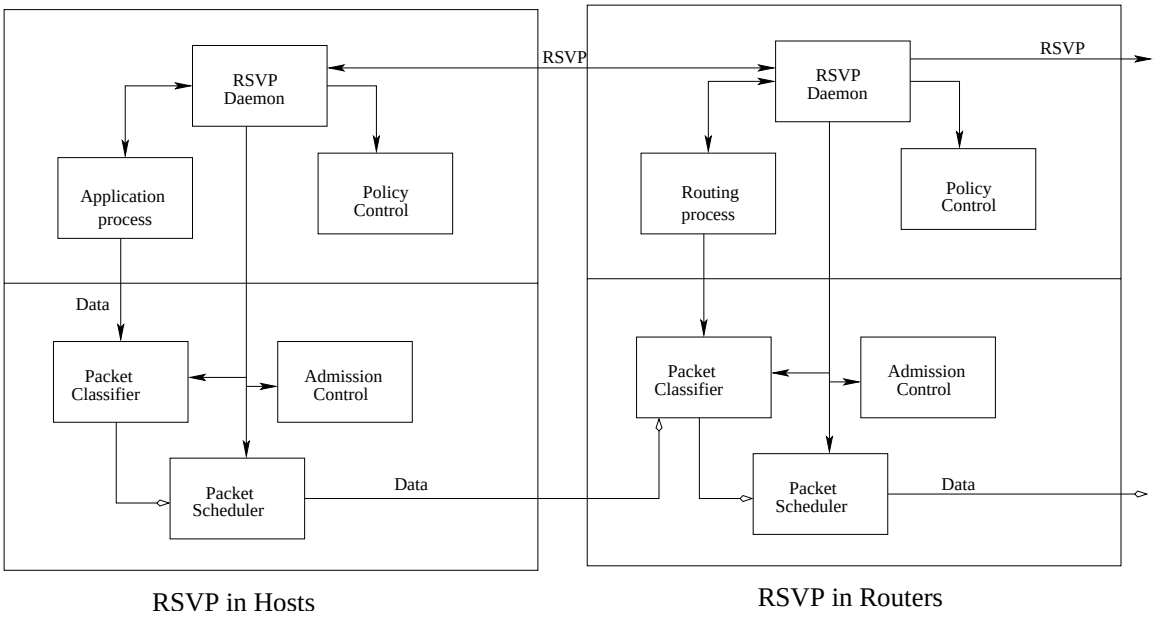


Figure 2.1: Traffic Control in RSVP [2]

sends it to the packet scheduler. The packet scheduler schedules the packet in a specific link layer interface to provide the requested QoS (flow-spec). Figure 2.1 illustrates the four components at the end host and in the routers.

2.2.2 Key features of RSVP

As mentioned earlier, in RSVP, the receiver end system initiates a reservation request, which makes RSVP scalable for a large number of receivers. The reservation request travels until the first multicast router where a reservation for the same session exists from a different receiver. Here, the new request is merged with the already existing

reservation and the merged request is sent to the sender. The merged request has a flow-spec that is the larger of the two requests being merged [4].

Soft state refers to the reservation state in routers and end nodes that will time-out if the refresh message is not received within a particular time period. It is the responsibility of the end nodes in an RSVP session to send RSVP refresh messages periodically to update the reservation state. The reservation maintenance is achieved through the same RSVP control messages that have been used to establish initial setup for reservation. Since RSVP resource maintenance and initial setup are done using the same RSVP control messages, the RSVP reservation state automatically adapts itself to routing changes, multicast group membership changes, and reservation modifications [4]. If the sender or receiver wants to change the existing reservations, it can be simply done by sending the same RSVP control message that was used to maintain the reservation state, but with new reservation parameters [4].

The Integrated Service architecture was defined to provide a set of extensions to the best effort traffic model currently used in the Internet. The framework was designed to provide special handling for certain types of traffic and to provide a mechanism for the application to choose between multiple levels of delivery services for its traffic [8]. A receiver may decide not to make any reservations and expect best effort service. That is why reservation and routing are independent of each

other. However this separation of reservation from routing could lead to choosing a path where resources are not available whereas there could be a alternate path with sufficient resources.

2.2.3 RSVP Messages

Applications using RSVP initiate connections by sending RSVP messages. There are seven RSVP messages, two among them being the primary ones (*PATH* and *RESV* messages).

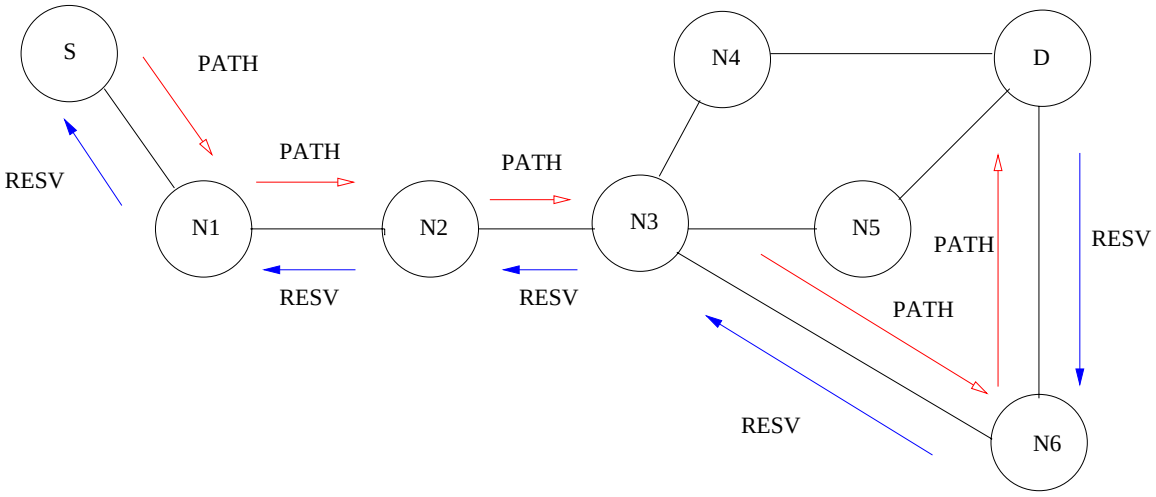


Figure 2.2: RSVP Messages

PATH Message

Whenever an RSVP end host wants to send data, it sends a *PATH* message to the destination. The *PATH* message creates a path state in each node it visits and stores the unicast address of the node that forwarded it to this node. This information is used by the *RESV* message while traveling upstream towards the sender. Each *PATH* message contains a *Sender TSpec* and an *Adspec*. *PATH* messages are also sent by the sender periodically to keep the path state alive in each of the nodes present in the predetermined path.

Sender TSpec describes the traffic the sender is expected to generate. The TSpec is taken into consideration when a decision is made regarding how many resources need to be reserved for a particular session. This prevents over-reservation of resources and unnecessary admission control failures.

Adspec measures the properties of each node along the path towards the destination [4]. These properties include QoS control capability, per-service characterization parameter values for each supported service, ability to support Integrated Service architecture, etc [21]. At each node, Adspec is updated by the traffic control module of that node and then sent to the next hop node. If a QoS control capability is not supported by an intermediate node, a flag is set in the Adspec for that QoS control capability to inform the receiver. This flag bit in the Adspec will notify the receiver

that its packet will receive best effort service in at least one node in the established path between sender and receiver.

RESV Message

Upon receiving the *PATH* message, the receiver interprets the sender's TSpec and Adspec, and then sends a reservation request using *RESV* message to reserve resources along the reverse path leading to the sender. The *RESV* message uses the previous hop information (PHOP) in the path state to find out its next hop node. The *RESV* message maintains the reservation state in each node it visits [4]. *RESV* messages are sent periodically from the receiver to keep the reservation state alive in each node of the path. This is referred to as a soft state; the nodes will automatically release the resources after a specific period of time as opposed to a hard state where a node will require an explicit reservation state tear message to release its resources. The resources are allocated in accordance with values of the flow-spec object.

PATHTEAR and *RESVTEAR* Message

PATHTEAR and *RESVTEAR* messages remove both the path state and reservation state in the node. The *PATHTEAR* message can be generated either by intermediate nodes due to a path state time out or by the sender when it wants

to close the session. It can also be sent when an application has completed data transmission. $PATHTEAR$ messages travel downstream towards the receiver. On comparable lines, $RESVTEAR$ messages can be generated either by the receiver or by any intermediate node due to a reservation state timeout. The $RESVTEAR$ message is sent upstream towards the sender.

$PATHERR$ and $RESVERR$ Message

$PATHERR$ or $RESVERR$ messages are generated when errors occur during the processing of the $PATH$ and $RESV$ messages respectively. The $PATHERR$ or $RESVERR$ messages are sent to the sender and receiver respectively. They do not change the state information in any of the intermediate nodes they visit.

$RESV\_CONF$

When a $RESV\_CONFIRM$ request is included in the $RESV$ message, each intermediate node sends a $RESV\_CONF$ message to the receiver, provided that the reservation of resources is successful in that node.

2.2.4 Non RSVP Clouds

RSVP messages can pass through non RSVP nodes on their way to the sender or receiver. However no state is maintained in these nodes. All packets of the current session will be serviced on a best effort basis in the RSVP unaware node. In particular, this may negatively affect the performance of an application requesting QoS because resource guarantees cannot be provided in RSVP unaware nodes.

2.3 Summary

RSVP is a signaling protocol used to reserve resources in the network. An RSVP sender advertises its traffic specification ($TSpec$) to the network before it starts transmitting data. This information is carried by the $PATH$ message to the destination. The $PATH$ message creates a path state for this flow in all the intermediate routers present along the path to the destination. In addition to the $TSpec$, $PATH$ message also carries $Adspec$ that measures the properties of each router along the path to the destination. Upon the receiving the $PATH$ message, the RSVP receiver sends a $RESV$ message to reserve resources along the path followed by the $PATH$ message (in the reverse direction). When the $RESV$ message reaches the sender, it starts transmitting data. RSVP uses a soft state approach to release the allocated resources.

Reservations have to be periodically refreshed to keep them alive otherwise the resources will be released by the network. Resources can also be explicitly released by end nodes when they finish their data transmission.

When sufficient resources are not available in the path followed by the *RESV* message, the call is dropped. However this does not necessarily mean that resources are not available in the network. In fact, resources may be available in the network but not contiguously along the path requested. This phenomenon is called resource fragmentation. Resource fragmentation can be reduced by dynamically redistributing resources of existing flows to accommodate the new flow. Chapter 3 explains resource fragmentation with respect to RSVP and the algorithms used to redistribute resources.

CHAPTER 3

DYNAMIC RESOURCE REDISTRIBUTION

3.1 Introduction

As explained in chapter 2, an RSVP source starts sending data only after its request for resources is granted by the network. That is, it starts transmitting data only after it receives the $RESV$ message from the receiver. If sufficient resources are not available in the predetermined path during the reservation ($RESV$) phase, the call is dropped. However, this does not mean that resources are not available in the network. In fact, the required resources may be available in the network, however not contiguously, along the path requested. At this point of time, we may be able to dynamically redistribute the resources, thus succeeding in granting the current QoS request. In this chapter, we introduce and explain our algorithm for the dynamic redistribution of resources in the network.

Consider the small section of a network in Figure 3.1. Assume that a call (flow1) is active in the path ABCDE. Now a call request (flow2) originates at node A for destination F. An RSVP $PATH$ message traverses downstream whereas an RSVP $RESV$ message traverses upstream towards the sender. Further assume that when

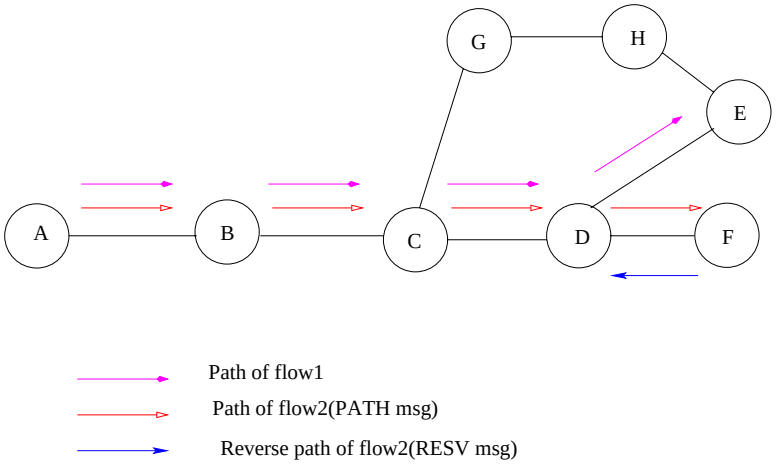


Figure 3.1: RSVP

the *RESV* message reaches node D, the QoS request cannot be satisfied due to the lack of resources along the path A-F. Here, instead of dropping the current call, if node D releases a certain amount of resources and redistributes it on other paths, it might be able to grant the requested QoS for flow1. Specifically, if node D reroutes flow1 through node G , it can release the resources associated with flow1 and give them to flow2. It is imperative that the entire process of redistributing resources should not affect the data flow of flow1 or any other flow in the network. This can be accomplished by exchanging messages between the nodes concerned. To facilitate resource redistribution, a flag is associated (rd_flag) with each flow that has been granted resources by the network. This flag determines whether resources of a particular flow can be redistributed or not. This flag is required to ensure that a flow is

not attempted for redistribution more than once by the network at the same time.

3.2 Special Messages

Some extra messages need to be incorporated into RSVP to facilitate the dynamic redistribution of resources. There are a total of eight extra messages that are used in our algorithm, henceforth referred to as special messages.

Consider a scenario when a connection fails at node N. To dynamically redistribute some of the resources in node N and free them in order to be able to allocate them to the failed connection, an alternate path needs to be established for the existing routes to which these resources are currently allocated. This alternate path must not form any loops nor should it include node N. Therefore, some flows from node N that may free the required resources if their resources are successfully redistributed are selected. The previous hops of these flows now need to find new paths to the destinations such that node N is not included in the path. The special messages are used in these previous hops of selected flows to establish new routes. While explaining the messages, the term previous hop of node N is used to refer to the previous hop of node N for a particular flow.

Resource redistribution cannot be accomplished if there is only one path to a destination. Therefore each node in the network contains a multiple next hop routing

table that stores multiple routes to the same destination. In this implementation, the number of routes available to a particular destination from any node is restricted to 4. The term route refers to routing information required to forward a message to a particular destination. In this case, the routing information is the next hop node number present along the path towards the destination.

S_TRIGGER Message

When a resource request fails at a node, the *S_TRIGGER* message is sent to the previous hop of the selected flow to trigger the *S_PATH* message.

S_PATH Message

The *S_PATH* message explores new routes to the destination. It examines the routing table at each node that it visits until it reaches the destination, to check if the next hop of this node can be added to the new path. The next hop of this node qualifies to be added to the new path only if it does not form a loop, or if it is not node N. If none of the next hops in the routing table qualify to be the next hop to the destination, the *S_PATH* message fails and does not proceed further. No attempt is made to find an other alternate path.

Once the next hop is successfully selected, the *S_PATH* message establishes a

special path state (SPS) in that node. This SPS (Figure 3.2) is used in the routing of data and consists of the *flow-id* which uniquely identifies a flow that passes through a node, the *phop* information that identifies the previous hop node, the *route number* for the next hop, the *destination node* for this flow, the *list of nodes* traversed so far and a *flag* indicating whether a path state exists or not. The route number is the appropriate index to the next hop entry in the multiple next hop routing table at the node. Information of the destination node is also included in the SPS. The SPS contains a complete list of all the nodes in the path so far, and a flag that is used to indicate that a path state already exists or does not exist. This is useful in determining whether the node is on the new path (path state does not already exist) or whether it is a common node on the old and new paths to the destination (path state already exists). If the path state does not already exist, a temporary path state is created, with the same information contained in a regular path state. No action is taken if a path state already exists in a node. When the S_PATH message reaches the destination, the destination sends a S_RESV message to the sender of the S_PATH message.

In Figure 3.2, an S_PATH message is sent from node 1 if a connection has failed at node N. There are four possible next hop entries in the routing table of node 1 for destination node 6. As node N is an invalid selection, node 2 is selected as the next

hop. At node 2, the possible next hops as given by the routing table are nodes N, 1, 3 and 13. Node N is clearly an invalid choice since it is in this node that the resource request failed, and node 1 is also not a possible selection because inclusion of node 1 in the path will form a loop. Therefore, node 3 is selected as the next hop. At node 3, 4 is the next hop, and at node 4, the next hop in the routing table is 5. At node 5, the next hop is 6, and although node 5 was in the old path too, it is selected as it does not present any loops.

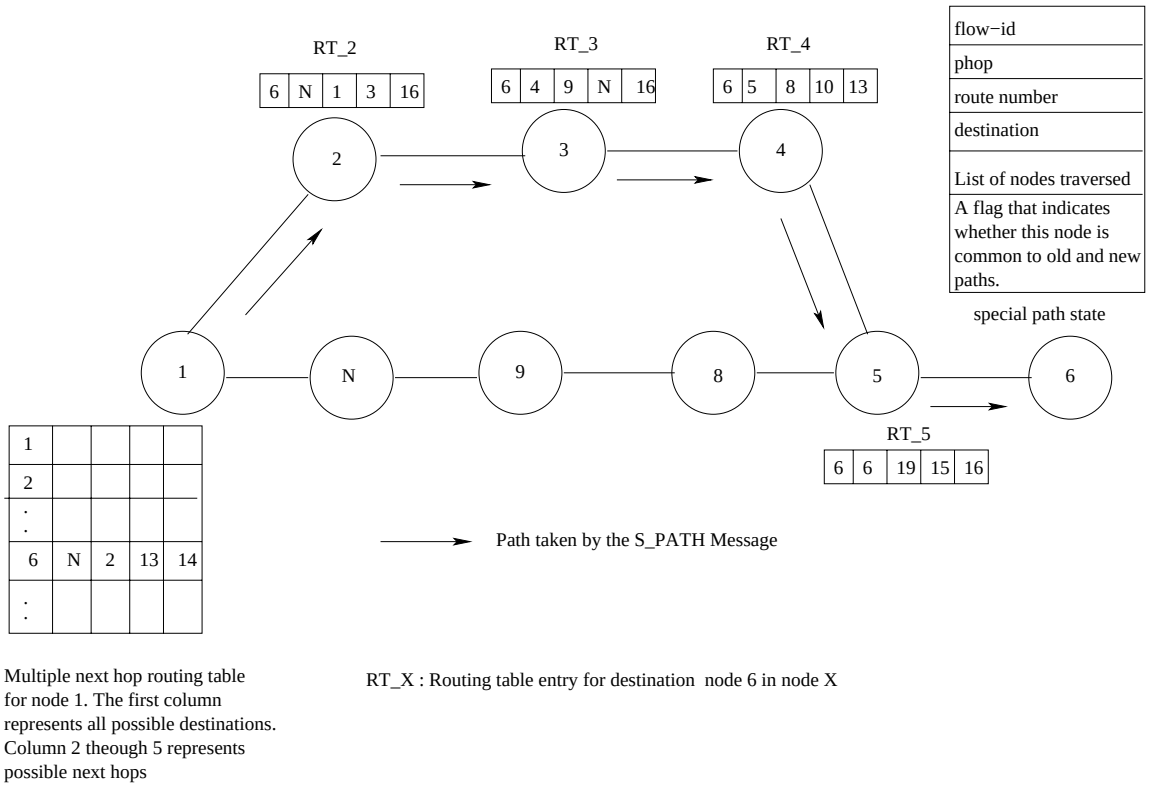


Figure 3.2: *S_PATH* Message

S_RESV Message

The *S_RESV* message is sent by the destination node along the reverse path of the *S_PATH* message to the node from which the *S_PATH* message originated, in order to reserve resources for the flow. The *S_RESV* message establishes a special reserve state (SRS), in the nodes it visits. If the node was already a part of the old path, it has the required reservation of resources and we set a flag in its reserve state to indicate that its resources cannot be redistributed. However, if the node does not already contain a reserve state, a special reserve state is created, containing the *flow-id*, uniquely identifying the flow and the Quality of Service requested (*qos*).

If resources are not available to be reserved, the *S_RESV* message fails, and a *S_RTEAR* message is sent to the destination to release the resources. No further attempt is made to allocate resources on an other alternate path. If the *S_RESV* successfully reaches the previous hop of node N, an *S_CONFIRM* message is sent by the *phop* node to the destination along the new path.

For example, in Figure 3.3, the *S_RESV* message travels the path 6-5-4-3-2-1. Since nodes 6 and 5 are common to old and new paths, no reservation is made in these nodes.

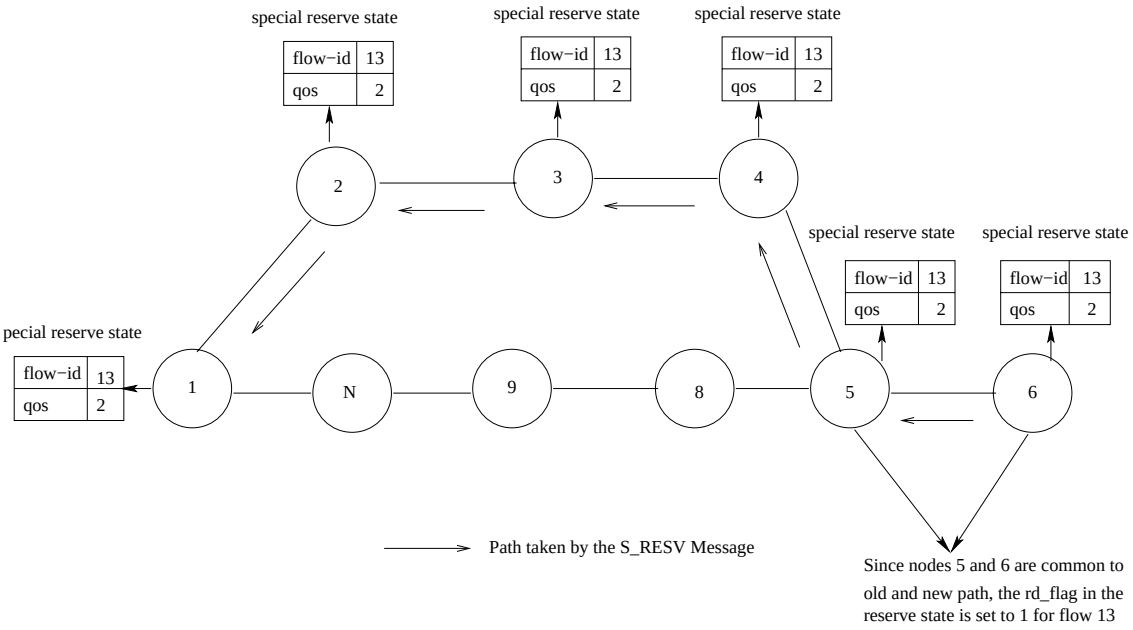


Figure 3.3: S_RESV Message

$S\_CONFIRM$ Message

The $S\_CONFIRM$ message is sent to confirm the reservation of resources along the new path to the destination, and to change the path states in the common nodes to reflect the new previous hops and next hops to the destination along the new path. Until $S\_CONFIRM$ reaches them, the common nodes in the old and new paths retain their path state, with the old previous hop node and old history of path to the destination. This information is unchanged to ensure that in the event of the failure of the S_RESV message, the old path is followed. However, if the S_RESV message is

successful, the new path needs to be followed and the path state information needs to be updated in these nodes. Before updating the previous hop of the node, the old *phop* is saved so that it can be used by the S_TEAR message. When the $S\_CONFIRM$ message reaches the destination, an S_TEAR message is sent to release resources along the old path in the reverse direction.

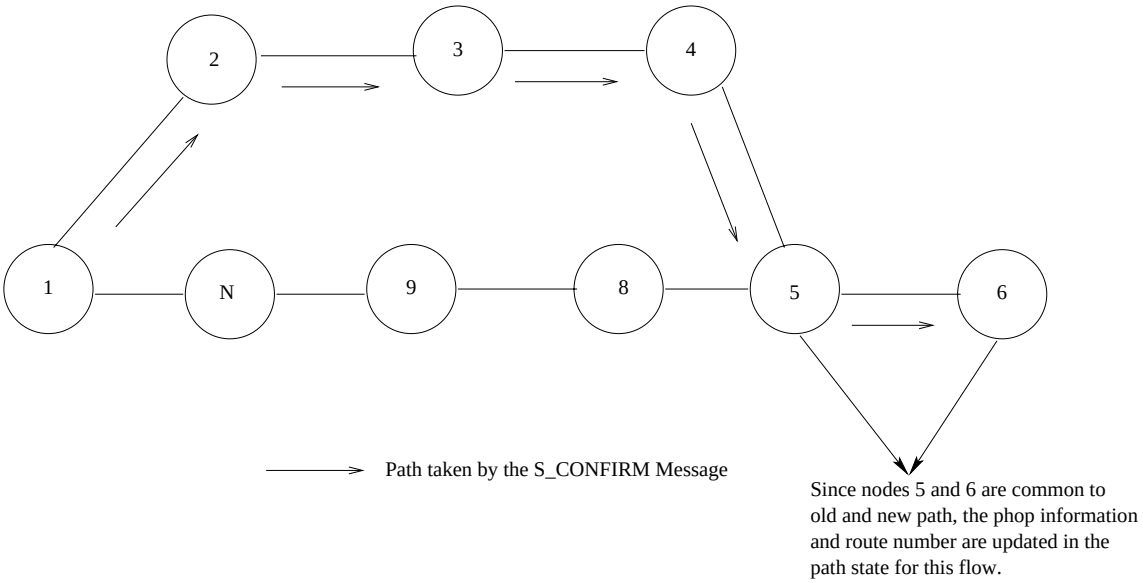


Figure 3.4: $S\_CONFIRM$ Message

As shown in Figure 3.4, the $S\_CONFIRM$ message traverses the nodes 1-2-3-4-5-6. Since node 5 is common to the old and new paths, this message changes the previous hop information in the node 5 to reflect the new path. In this example, it does not change the next hop information in node 5 because the path to node 6 in

the new and old paths is the same.

S_REFRESH Message

In RSVP, refresh messages are sent periodically to maintain the reservation state. If the nodes do not receive a periodic refresh message, they release their resources. The *S_REFRESH* message is sent by the previous hop of node N, along the old path so that resources are not released prematurely before being allocated along the new path. The previous hop changes its next hop to reflect the new path, and therefore any messages, including the refresh message must follow this new path to the destination. In the event that the new path is longer than the old path, messages might take longer to reach the destination along the new path, but if the destination or any other node present in the old path does not receive a refresh message, it may release its resources, as do the common nodes along the old and new paths to the destination. This results in there being no complete path to the destination. To keep the resources reserved along the old path to the destination, the *S_REFRESH* message is sent.

In the Figure 3.5, assume that a flow (f1) fails at node 3. We select a flow f2 and try to redistribute its resources. After the node 2 receives the *S_RESV* message, any messages and data concerned with flow f2 follow the path 1-2-7-8-9-10-5-6. After receiving the *S_RESV* message, if node 2 does not send a *S_REFRESH* along the

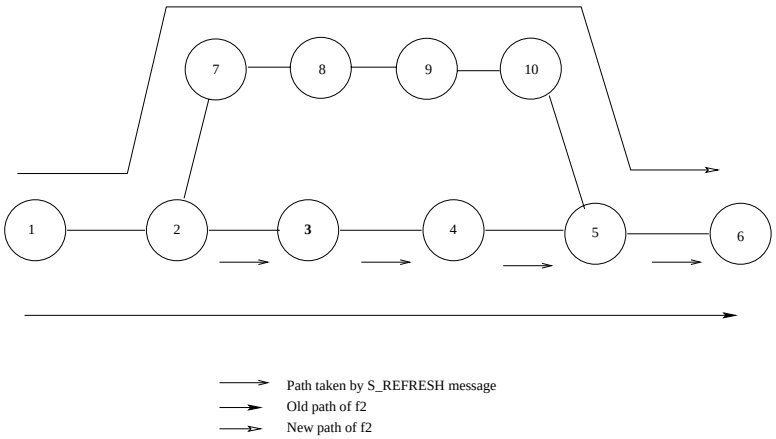


Figure 3.5: *S_REFRESH* Message

old path to nodes 3, 4, 5, and 6 to retain their reservation of resources, they may release them. The regular reserve message and any other message of that flow will traverse the new path along nodes 7, 8, 9, 10, 5 and 6 from node 2. As this is a longer path, the refresh message may not reach in time to keep the path and reservation state alive, causing the resources to be released in nodes 3, 4, 5 and 6. When resources are released in nodes 5 and 6, no complete path exists to the destination node 6 for this particular flow. The *S_REFRESH* message guards against this by retaining resources along the old path in nodes 3, 4, 5 and 6.

NOTIFY_SUCCESS Message

The *NOTIFY_SUCCESS* message is sent by the previous hop of node N to notify node N of the successful redistribution of resources. There may be more than

one $NOTIFY\_SUCCESS$ message, depending on the *qos* of the connection that has failed at node N. Upon receiving all the $NOTIFY\_SUCCESS$ messages, indicating that the node N has all the required resources for the new connection, it allocates these resources to the failed connection.

S_TEAR Message

The destination node sends the S_TEAR message along the reverse old path to release the resources reserved for the flow. This is done only after the destination receives the $S\_CONFIRM$ message. The previous hop of the node is checked in the path state, as is the prior existence of the path state. If the path state was already present, the S_TEAR message cannot be sent to the present previous hop of this node, as this actually represents the new path. Therefore, it checks the saved phop (during the $S\_CONFIRM$ message), and sends the S_TEAR to this phop to release the resources.

S_RTEAR Message

The S_RTEAR message is sent to the destination node from the node where the S_RESV message has failed. If resources cannot be allocated along the reverse new path, the S_RESV message fails. This triggers the S_RTEAR message, that

traverses the new path to the destination node, releasing resources that have already been reserved along the way. If any of the nodes along the path are nodes that are common to the old and new paths, resources are not released, and the S_RTEAR continues till the destination.

3.3 Algorithms

Redistribution of resources broadly involves two steps: *selecting* the flows whose resources will be redistributed, and *redistributing* the resources of the selected flows without affecting existing flows in the network. This section describes the algorithms used to accomplish the above two steps.

Consider a network of N nodes. Let F be the total set of flows in the network. Let F^j be the total set of flows passing through node j such that $F^j \subseteq F$ and $f_x^j \in F^j$ where $1 \leq x \leq |F^j|$. Let T be the set of possible *qos* requests. We have considered four classes of resource requests in terms of buffer space, i.e., $T = \{1, 2, 4, 8\}$. Let *qos* be the resource request of flow f_y^j. If resource request for this flow fails at node j, node j tries to free resources for the flow f_y^j using the Dynamic Resource Redistribution algorithm explained in this section.

3.3.1 Flow Select Algorithm

When the resource request for a flow fails at a node, a certain number of other flows are selected to be redistributed from this node, to free the resources for the failed flow. This is done by implementing either of two different algorithms in the following order

1. Best Fit Multiple Flow Select Algorithm (MFS)

2. Best Fit Single Flow Select Algorithm (SFS)

The MFS algorithm is called first in an attempt to redistribute resources of multiple flows each with a smaller quantity of resources requested than that of the failed flow. For example, if a flow with a resource request for 4 units of buffer space fails, we try to redistribute other flows that have resource requests less than four units each, totalling 4 units. That is, flows with resources of 2 units or 1 unit each of buffer space are selected. We take this approach as the redistribution of flows with smaller resource requests is more likely to succeed than that of flows with larger resource requests.

If the MFS algorithm fails, we call the SFS algorithm to find a single flow with at least the same resource requests as that of the failed flow, and redistribute its resources.

The possible classes of resource requests (buffer space requested) considered in this algorithm are 1, 2, 4 and 8 units. Therefore the flows that can be selected to be redistributed for each of the above classes are:

When the failed flow's resource request is 8 units, we attempt to select flows for redistribution in the following order: Using MFS, 2 flows with resources of 4 units each, or 4 flows with resources of 2 units each, or 8 flows with resources of 1 unit each. Using SFS, 1 flow with resources of 8 units.

When the failed flow's resource request is 4 units, we attempt to select flows for redistribution in the following order: Using MFS 2 flows with resources of 2 units each, or 4 flows with resources of 1 unit each. Using SFS, 1 flow with resources of 4 units, or 1 flow with resources of 8 units.

When the failed flow's resource request is 2 units, we attempt to select flows for redistribution in the following order: Using MFS 2 flows with resources of 1 unit each. Using SFS, 1 flow with resources of 2 units, or 1 flow with resources of 4 units, or 1 flow with resources of 8 units.

When the failed flow's resource request is 1 unit, we attempt to select flows for redistribution in the following order: Using SFS, 1 flow with resources of 1 unit, or 1 flow with resources of 2 units, or 1 flow with resources of 4 units, or 1 flow with resources of 8 units.

The Flow Select Algorithm is as follows (Algorithm 1):

Algorithm 1

Let FLOWS be an empty set.
if(qos == 1)
    FLOWS = Single_Flow_Select(qos)
else
{
    FLOWS = Multiple_Flow_Select(qos)
    if(|FLOWS| < 1)
        FLOWS = Single_Flow_Select(qos)
}
return FLOWS

3.3.2 Best Fit Multiple Flow Select Algorithm

In this algorithm, we first try to minimize the number of flows being redistributed at a node. For example, if a resource request of the failed flow is 4 units, we can redistribute either two flows with resource requests of 2 units each, or four flows with resource requests of 1 unit each. We try to select two flows with resources of 2 units each for redistribution before attempting to select the latter. Thus the MFS algorithm attempts to select fewer flows to reduce the extra traffic generated in the network because of the resource redistribution attempt.

Let Max be a function that takes a set as a parameter and returns the largest element from the set.

Best Fit Multiple Flow Select Algorithm is as follows (Algorithm 2).

Algorithm 2

$P = \{x | x \in T \bigwedge x < qos\}$
$f(n) = qos \div n$
flag = false
$MFLOWS = \{\}$
while(|P| > 0 and !flag)
{
    a = Max(P) and b = f(Max(P))
    Let (a,b) be a two tuple element selected to search F^j *for 'b' number of flows with a resource allocation of 'a' units.*
    search_success = false
    search_count = 0
    $i = 1$
    while($i \leq |F^j|$ *and !search_success)*
    {
        if($f_i^j.qos == a$ *and* $f_i^j.rd\_flag == 0$*)*
        {
            search_count = search_count+1
            if(search_count == b)
                search_success = true
        }
        i =i+1
    }
    if(search_success)
    {
        search_complete =false
        search_count=0
        i=1
        while($i \leq |F^j|$ *and !search_complete)*
        {
            if($f_i^j.qos == a$ *and* $f_i^j.rd\_flag == 0$*)*
            {
                $f_i^j.rd\_flag$ *=1*
                $MFOWS = MFLOWS \cup f_i^j$
                search_count = search_count+1

```
                if(search_count == b)
                {
                    search_complete = true
                    flag=true
                }
            }
            i=i+1
        }
    }
    P = P − {a}
}
return MFLOWS
```

3.3.3 Best Fit Single Flow Select Algorithm

A single flow with at least the same resources as that of the failed flow is selected to be redistributed in this algorithm. This means that for every failed flow, we try to find a single flow of the same or higher resource amount to be redistributed. For example, if a flow with a resource request of four units fails, and the MFS algorithm is also unsuccessful, an attempt is made to find a single flow with resources of 4 units to be redistributed. If this also fails, we try to find a single flow with resources of 8 units.

Let Max be a function that takes a set as a parameter and returns the largest element from the set.

The Best Fit Single Flow Select Algorithm is as follows (Algorithm 3):
Algorithm 3

$P = \{x | x \in T \bigwedge x \geq qos\}$
flag = false
$SFLOW = \{\}$
while(|P| > 0 and !flag)
{
    a = Max(P)
    Let a be an element selected to search F^j *for a*
    single flow with a resource allocation of 'a'.
    search_success = false.
    $i = 0$
    while($i \leq |F^j|$ *and !search_success)*
    {
        if($f_i^j.qos == a$ *and* $f_i^j.rd\_flag == 0$*)*
        {
            $f_i^j.rd\_flag = 1$
            $SFLOW = SFLOW \cup f_i^j$
            flag=true
        }
        i =i+1
    }
    $P = P - \{a\}$
    }
return SFLOW

3.3.4 Dynamic Resource Redistribution Algorithm

Definition of terms used in the Dynamic Resource Redistribution (DRR) algorithm

A *forward path* is defined as the path, the direction of which is from the sender to the receiver of the flow. when the direction of the path is from receiver to sender, it is known as a *reverse path.*

Loop: Let L_s be the set of all nodes in the path from the sender to the previous hop of the current node. If the next hop from the current node is any of the nodes in the set L_s, the path is said to form a loop.

The DRR algorithm is as follows(Algorithm 4):

Algorithm 4

1. *Let S be a subset of F^j consisting of flows satisfying Flow Select algorithm. Let $i = |S|$, and $S = \{f_1^j ... f_i^j\}$. If i is zero, exit the algorithm and drop the flow f_y^j, otherwise goto 2.*

2. *A timer T1 is started at Node j for the successful notification of redistribution of resources by each of the flows in set S. A counter C is initialized to i.*

3. *For each flow in set S, f_k^j, where $1 \leq k \leq i$, the following steps are carried out:*

   (a) *Let $PHOP_k$ be the previous hop of Node j along the path of flow f_k^j. An $S\_TRIGGER$ message is sent to $PHOP_k$ from node j, to trigger the S_PATH message used in redistribution of resources.*

   (b) *From $PHOP_k$ an S_PATH message is sent to the destination D_k of flow f_k^j, along a new path P_k. This new path is determined in such a way that it does not contain node j, nor any loops.*

(c) *After receiving the S_PATH message, node D_k sends an S_RESV message to $PHOP_k$ along the reverse path of P_k. If the S_RESV message fails before reaching the node $PHOP_k$, an S_RTEAR message is sent to node D_k along the forward path P_k.*

(d) *When the S_RESV message reaches the node $PHOP_k$, node $PHOP_k$ does the following:*

i. *It sends an $S\_CONFIRM$ message to the node D_k along the new path P_k.*

ii. *It sends an $S\_REFRESH$ message along the old path of flow f_k^j to node D_k.*

iii. *It changes its route_number field in the path state to reflect the new path along P_k to node D_k.*

iv. *Node $PHOP_k$ finally sends a $NOTIFY\_SUCCESS$ message after a specified period of time, to Node j to indicate successful redistribution of resources for flow f_k^j. The delay for the $NOTIFY\_SUCCESS$ message is introduced so that any messages traversing the old reverse path from destination D_k are not lost.*

(e) *Upon receiving the $S\_CONFIRM$ message from $PHOP_k$, node D_k sends*

an S_TEAR message to node j along the old path of flow f_k^j to release the resources. It also changes its own PHOP field to reflect the new previous hop according to the reverse path of P_k.

(f) On receipt of a NOTIFY_SUCCESS message from $PHOP_k$,

i. If the timer has not timed out, node j decreases the counter C by 1. If C becomes 0, node j releases the resources it had allocated to flows in set S and re-allocates them to flow f_k^j.

ii. If the timer has already timed out when node j receives the NOTIFY_SUCCESS message for a flow, it removes all state information for that flow and the flow maintains its new path.

4. If the timer T1 at node j times out before it receives the NOTIFY_SUCCESS message from all the flows in set S, the flow f_y^j is dropped.

3.3.5 Illustration of the DRR Algorithm

Consider a part of the network as shown in Figure 3.6. Let a flow f_y^j with a resource request of 4 units of buffer space fail at node j. The subscript y identifies a unique flow in node j. Assume that the Flow Select algorithm selects two flows f_3^j and f_4^j

with resource allocation of 2 units of buffer space to redistribute. In our illustration, we show how we redistribute the resources of flow f_3^j. Resources of flow f_4^j can be redistributed similarly.

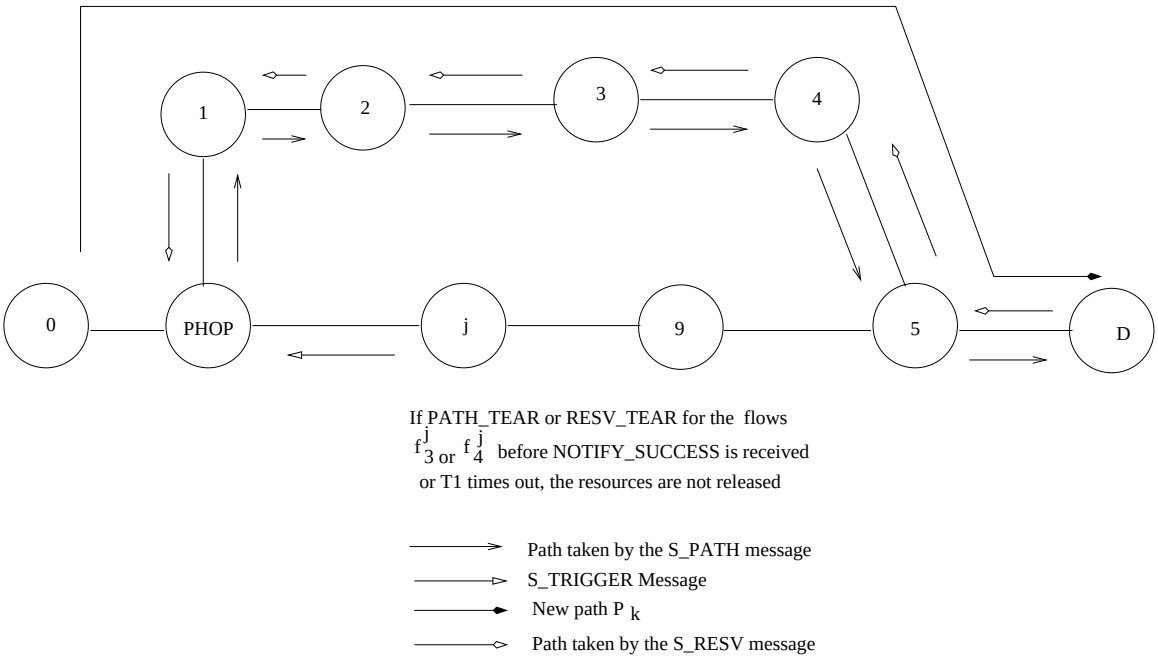


Figure 3.6: Example of Resource Redistribution

Figure 3.6 depicts a scenario where a resource request for a flow f_y^j fails at node j. Let PHOP and D be the previous hop node and destination respectively for the flow f_3^j. The figure shows the path taken by the $S\_TRIGGER$ and S_PATH messages. A timer T1 is started for the successful notification of redistribution of resources by the flows f_3^j and f_4^j. The $S\_TRIGGER$ message triggers S_PATH message in the

node PHOP. The S_PATH message explores a new path to node D. When S_PATH message reaches the destination, it sends the S_RESV message to PHOP along the reverse new path.

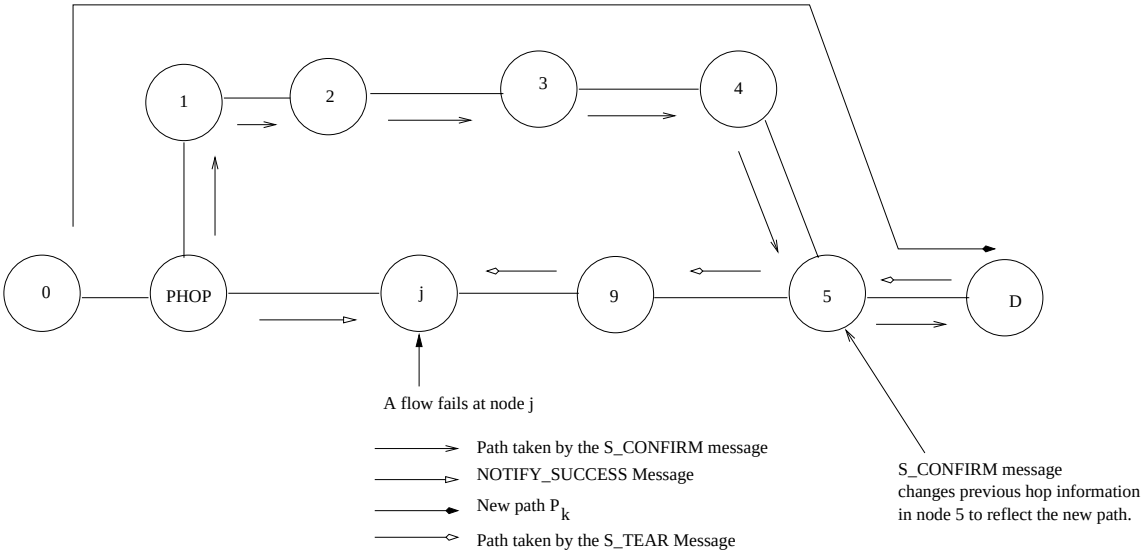


Figure 3.7: Example of Resource Redistribution

When the S_RESV message reaches the PHOP node (refer Figure 3.7), the PHOP node sends an $S\_CONFIRM$ message to node D, to confirm the resource reservation for the new path. The $S\_CONFIRM$ message also updates the path state of the nodes that are common to the old and new paths to reflect the new path that has been established. After sending the $S\_CONFIRM$ message, PHOP now sends an $S\_REFRESH$ message along the old path to refresh the path and reservation states. It then updates its next hop information in the path state to reflect the new path.

Finally the PHOP node sends a $NOTIFY\_SUCCESS$ message to node j after some period of time to indicate successful redistribution of resources for the flow f_3^j. In the meantime, when the $S\_CONFIRM$ message reaches the node D, it sends an S_TEAR message to release the resources along the nodes present in the old path. This message does not release resources in the nodes that is common to old and new paths. When node j receives $NOTIFY\_SUCCESS$ message, it checks whether the resources of flow f_4^j are already redistributed or not. If the resources of f_4^j are already redistributed, node j transfers the freed resources of flows f_3^j and f_4^j to the flow f_y^j. If node j is still awaiting the $NOTIFY\_SUCCESS$ message from flow f_4^j, no action is taken. If T1 times out before node j receives the $NOTIFY\_SUCCESS$ messages from the flows f_3^j and f_4^j, flow f_y^j is dropped.

3.4 Maintaining a path at all time

When an attempt is made to redistribute the resources of a flow on a different path (new path), the old path is not affected until its resources are successfully redistributed. In other words, at least one path remains active between the sender and receiver when this algorithm is applied to a flow unless and until it is explicitly torn down by the sender or its reservation state times out.

When the destination sends the S_RESV message to reserve resources to the node where the S_PATH message originated, a special reserve state is created in each node, as explained earlier. If this message traverses through a node that is common to the old and new path, no changes are made to the *phop* field of the path state present in that node. Thus it is guaranteed that if the S_RESV message fails before reaching its intended receiver node, the old path is preserved.

When the S_RESV message reaches its intended receiver, it sends an $S\_REFRESH$ message along the old path to the destination, to keep the old path and reserve states alive. If the new path is longer than the old path, the path state and the reserve state of nodes on the path, including the common nodes on the old and new paths may time out if they do not receive the refresh message on time. Therefore an explicit refresh message is sent to keep the path and reservation states alive.

When the S_RESV message fails due to the unavailability of resources, the S_RTEAR message is sent to the destination to release the resources allocated so far. Resources are not released in the nodes that are common to the old and new path, as this would mean loss of resources, not only for the new path, but for the old path as well. Thus, the old path is maintained.

The $NOTIFY\_SUCCESS$ message is sent to node j after a specific delay of time. This delay is introduced to allow the $S\_CONFIRM$ message to reach the destination and confirm a new path to the destination. If there is no delay in sending the NOTIFY_SUCCESS message and the destination is not yet aware of the new path, it sends a $RESV$ message to maintain the reservation state along the old path. Upon reaching node j, the $RESV$ message may fail if the node has already received all the $NOTIFY\_SUCCESS$ messages and removed its reserve state.

3.5 Special Cases

If a $PATH\_TEAR$ or $RESV\_TEAR$ message for one of the flows being redistributed is received at node j, the resources reserved for that flow at node j are not released, as they are being held for the new flow. Node j waits until it receives a $NOTIFY\_SUCCESS$ message for all the flows that are being redistributed or until it times out before releasing the resources to the new flow.

After sending the S_PATH message, the reserve state in the previous hop of the node may be explicitly deleted by the sender or it may time out. When the S_RESV message reaches its intended destination, it will not find a reservation in that node. In this case, an S_RTEAR message is sent to the destination to release all the resources for the flow, as they are not required any more (refer Figure 3.8).

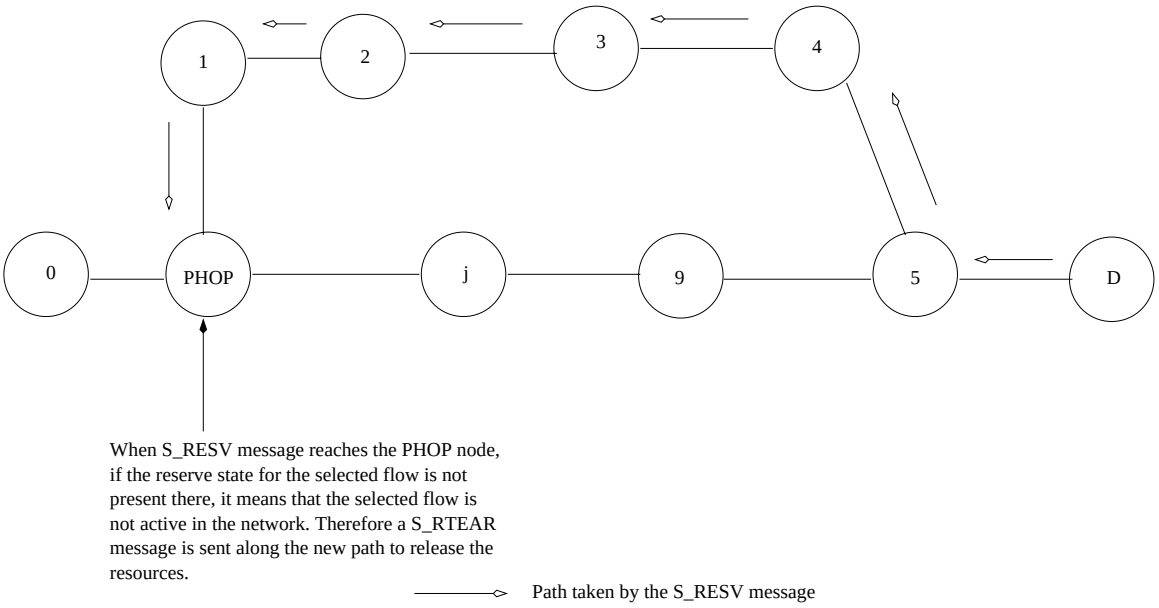


Figure 3.8: Special Case

3.6 Summary

Redistribution of resources constitutes the following steps: Selecting the flows whose resources will be attempted for redistribution, and the attempt to redistribute the resources of the selected flows without affecting the existing flows in the network. Two algorithms are presented in this chapter for flow selection and both of them were used, depending on what the resource demand was of the flow that failed due to resource unavailability. The main criterion of the flow select algorithm is to minimize the number of flows whose resources will be attempted for redistribution. This is due to the extra traffic introduced in the network by the attempt to redistribute

resources, and so that this overhead traffic does not congest the network. While selecting flows, we also try to choose flows with small resource allocations because they are more likely to succeed in redistribution compared to those flows with large resource allocation. After the flows are selected, special messages are exchanged between the nodes concerned to redistribute the resources.

In order to observe the performance of the above algorithms, several simulation experiments were carried out. The simulation model and the results of the range of experiments carried out is discussed in chapter 4.

CHAPTER 4

EXPERIMENTS AND RESULTS

4.1 Simulation Model

The main focus of this thesis is to evaluate the overall performance of unicast RSVP when Dynamic Resource Redistribution (DRR) is incorporated in it. An object oriented event based simulator (NRLSIM) is used to measure the efficacy of the DRR algorithm.

The NRLSIM is a basic simulation engine that can be used to simulate any event driven system. The type and the nature of the event is transparent to the simulator and can be custom defined by the model. In other words, the simulator acts as a black box (refer Figure 4.1), and can be used to simulate any event as required by the user.

The simulator consists of a *Simulator* class and an *Event* class. All the custom defined events are derived from the *Event* class. Whenever an event is serviced, control is transferred to the *ExecuteEvent* function of that specific event class. Therefore this function is declared as virtual in the *Event* class and is overridden by all custom

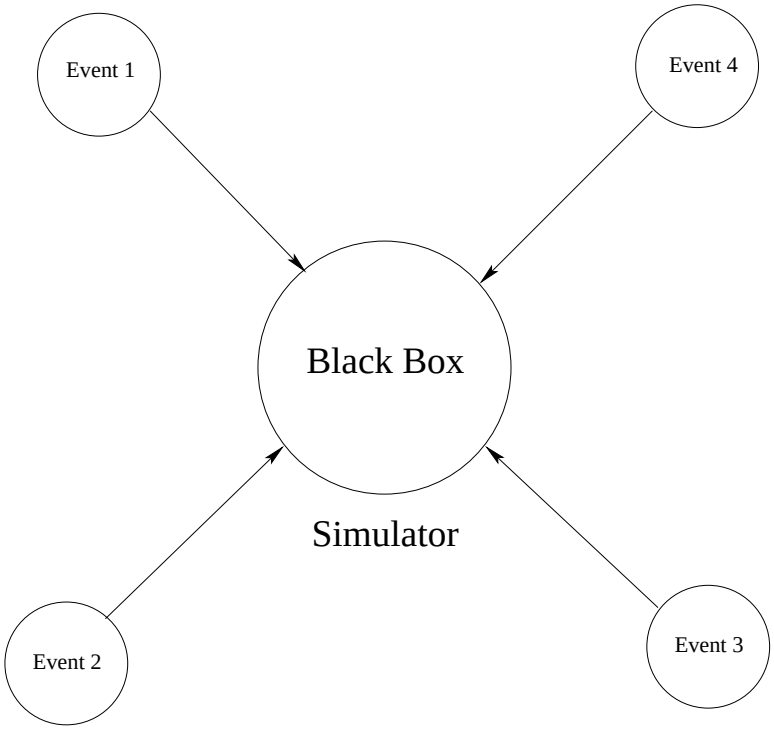


Figure 4.1: Simulation Model

defined events. Events are inserted into the event queue using the *InsertEvent* function, and removed from the event queue with the *RemoveEvent* function. Association between the events and the simulator is brought about by using pointer variables in the *Event* class and *Simulator class*. The object model of these classes is shown in Figure 4.2.

Two types of events have been designed in the implementation - *NodeEvent* and *SpecialNodeEvent*. A *NodeEvent* defines the actions to be taken when a node receives

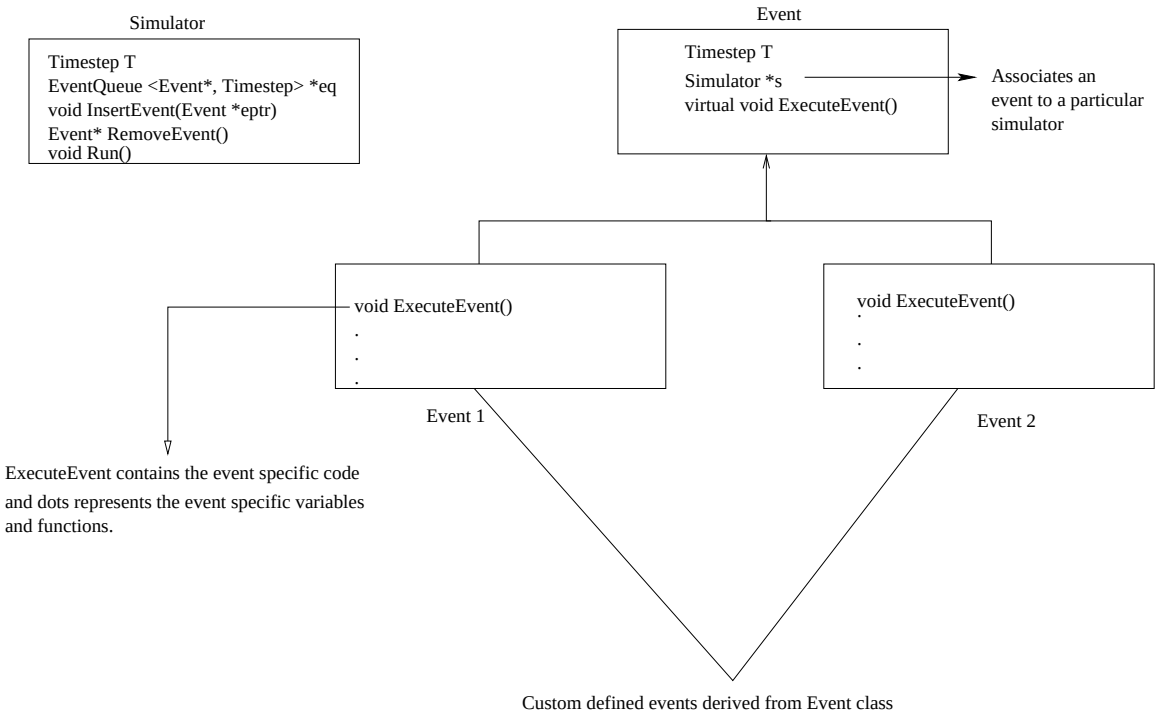


Figure 4.2: Object Model

an RSVP message, while a *SpecialEvent* defines the actions to be taken when a node receives a special message.

The state of a node refers to the collective state (values) of data structures associated with the node, such as path state, reserve state, buffer space, etc. These data structures may grow or shrink depending on events that occur in the node. An event may or may not change the state of the node. For example, upon receiving a $PATHTEAR$ message for a specific flow, the node may have to remove its path and reserve states if they already exist for that flow. Otherwise, no action is taken.

Traffic in the network is modeled in terms of flow requests with specific buffer requirements. A request is characterized by a source node, destination node, buffer requirements and the duration of this flow. The generation of requests is Poisson distributed with an average arrival rate of λ calls per minute. The duration of each flow is exponentially distributed with an average duration of D minutes. Buffer space requests are uniformly and non-uniformly distributed over the set $T = \{1, 2, 4, 8\}$.

The DRR algorithm was tested on different networks modeled by graphs of different sizes and average densities [1]. Every node in the network acts both as a router and end host. Links in all topologies have a fixed transmission delay of 3 ms and 2 ms for RSVP messages and special messages respectively. Each node in the network is assumed to be RSVP aware, i.e., it can process and forward RSVP and special messages. Messages sent over any link in the network are also assumed never to be lost or damaged. The network remains static throughout the duration of the simulator. Each node has a routing table with multiple next hop entries for a particular destination. In other words, there are multiple paths to a particular destination from a node. Dijkstra's shortest path algorithm is used to find the shortest path between nodes and complete the routing tables in all the nodes present in the network before starting the simulation of the algorithm.

[1] Average density of a network here refers to the average degree of a node in the network

4.2 Experimental Analysis

In all the experiments, the performance of RSVP with DRR (RSVP_DRR) was compared to conventional RSVP. There are three broad categories for comparisons - number of drops (request failure due to resource unavailability), resource utilization in the network, and the time complexity. In the time complexity experiment, successive attempts are made to accommodate a call in the network until its resource request can be granted by the network. Usually, unavailability of resources leads to a call being dropped in both RSVP_DRR and RSVP. The interpretation of unavailability of resources in RSVP_DRR is that the attempt(s) to redistribute the resources was also unsuccessful. Here, if a resource request fails, an attempt is made to allocate resources for the failed flow by re-trying the request until it is successful. The delay between each attempt to allocate resources is exponentially distributed with an average delay of μ minutes.

When the experiment is conducted on RSVP, all the request characteristics (source node, destination node, resource request, duration of flow and time of request) are stored and the same test set of connections is used while executing the RSVP_DRR. This ensures that all the experiments and comparisons are on exactly the same traffic pattern. Experiments were conducted on a 30 node network with varying average

densities of 3, 5 and 7. For each of these topologies, there were a set of experiments carried out. Experiments were conducted for two different distributions of QoS requests. In the first experiment, calls with QoS requests of 1, 2, 4 and 8 units of buffer space are generated with a probability of 50%, 25%, 15% and 10% respectively. In the second experiment, calls with QoS requests of 1, 2, 4 and 8 units of buffer space are each generated with a probability of 25%.

Different average call holding times were used for the experiments. The comparisons between RSVP_DRR and RSVP are presented for different call holding times, in terms of total number of calls dropped and resource utilization in the network. The results for the number of calls dropped are further categorized in terms of the requested QoS. As mentioned in the previous chapter, four different classes of QoS requests were considered in the experiments conducted. Results for the calls dropped are therefore presented as a total for different call holding times, and also as the number of calls dropped for each value of QoS (1, 2, 4, 8) for a single hold time.

4.3 Non-Uniform QoS Request Distribution

In this experiment, calls with QoS requests of 1, 2, 4 and 8 units of buffer space are generated with a probability of 50%, 25%, 15% and 10% respectively. The number of calls dropped in RSVP_DRR and RSVP were measured and compared, in order

to evaluate the performance of RSVP_DRR. The results confirm that fragmented resources in the network increase the number of calls dropped in conventional RSVP, and that the redistribution of resources reduces the degree of fragmentation in the network.

When the average density of the network is 5, RSVP_DRR performs compares favorably to RSVP in terms of number of calls dropped. Initially, there are no dropped calls in the network, as shown in Figure 4.3. After a significant number of calls have been established, resources may become unavailable for new incoming calls. Incoming calls are dropped when their resource request cannot be granted by the network. At this point, a steep increase in call drops is observed due to resource unavailability. However, the slope of the curve that characterizes the number of calls dropped decreases after a period of time. This decrease is a direct consequence of resources becoming available in the network after the initial calls have been serviced.

The number of calls dropped increases as the average holding time of a call increases as can be seen in Figure 4.3. There is a marked difference in the number of calls dropped when the average call holding time is 5 minutes and when the average call holding time is 20 minutes. This is due to the fact that calls request and hold resources for a shorter duration, and thereafter release the resources within a short period of time. This greatly reduces the number of drops. The longer a call holds

resources, the less available resources there are in the network for that duration, and consequently higher are the number of calls dropped.

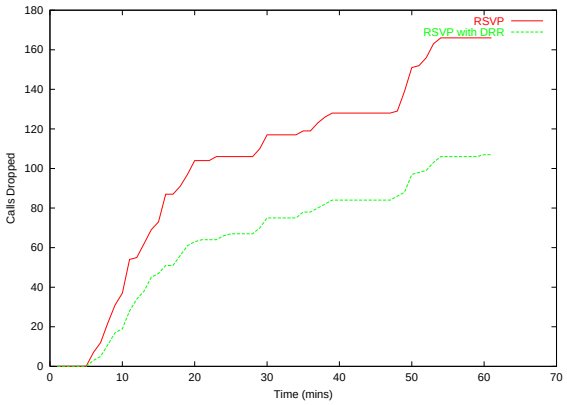
RSVP_DRR closely follows the performance of RSVP, but is nevertheless an improvement on it in terms of the number of calls dropped (Figure 4.3). When the average call holding time is short, RSVP_DRR is more effective than when the average call holding time is long, as resources are available sooner for redistribution. There is a decrease of 36% in the number of calls dropped in RSVP_DRR over RSVP, when the average call holding time is 5 minutes. As the average call holding time is increased, this percentage decreases (16% for 10 minutes, 9% for 15 minutes and 7% for 20 minutes) as resources are held longer by the calls before being released.

Figure 4.4 compares the total number of calls dropped for different QoS requests. This experiment determines the optimal QoS request for which resources can be redistributed effectively. This experiment was also useful in verifying the specific QoS request for which calls were denied resources in RSVP, as a consequence of resource fragmentation. An interesting observation is that the QoS requests determined in both RSVP_DRR and RSVP above are the same, indicating that calls that have been denied resources in the latter due to resource fragmentation, are the ones that are most easily redistributed using RSVP_DRR.

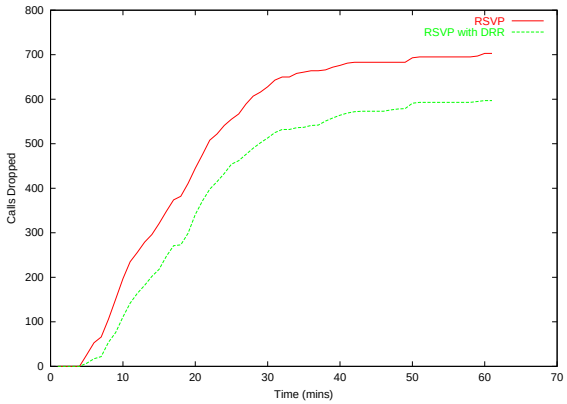
When the QoS request is low, fewer calls are dropped than when the QoS request is high (4 and 8). In particular, we observe that fewer calls are dropped with a QoS request of 1 unit, when compared with a QoS request of 8 units. This is because calls with lower QoS requests can be more easily accommodated in the network. Calls with small QoS requests can be redistributed more easily than those with high QoS requests.

Resource utilization is a measure of the percentage of used resources over total available resources in the network. Resource fragmentation leads to under-utilization of resources in the network. This experiment was conducted to investigate the effect of redistribution of resources on resource utilization in the network.

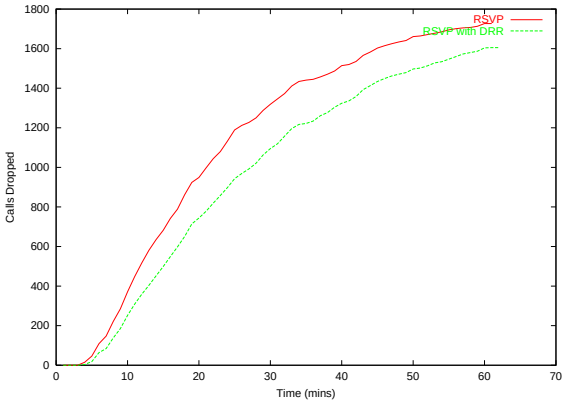
Resource utilization in the network is higher with RSVP_DRR than with RSVP, as seen in Figure 4.5. When the network succeeds in redistributing resources, more resources in the network are utilized. Further, when the average call holding time is long, resources of more calls will be redistributed than when the average call holding time is short. It can be concluded that redistribution of resources reduces resource fragmentation in the network, and thereby improves resource utilization in the network over conventional RSVP.



(a) Hold Time = 5 min

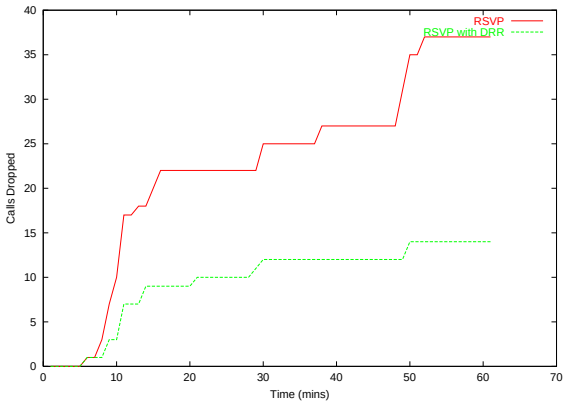


(b) Hold Time = 10 min

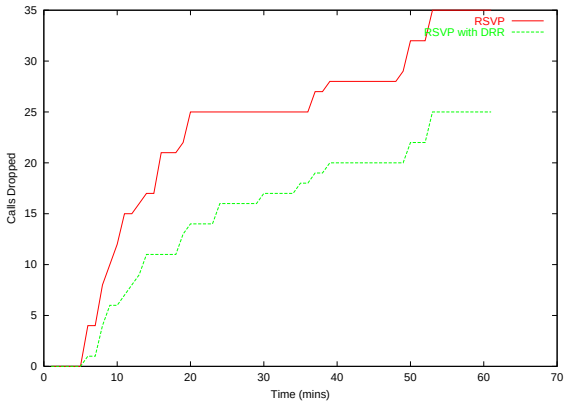


(c) Hold Time = 20 min

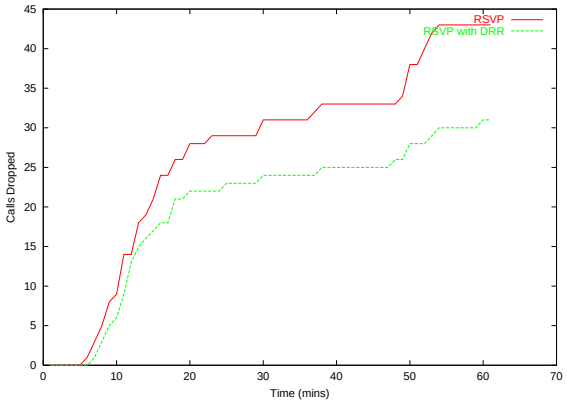
Figure 4.3: Drop Graph for a Network Size of 30 Nodes with an Average Density of 5 (Non-Uniform QoS Request Distribution)



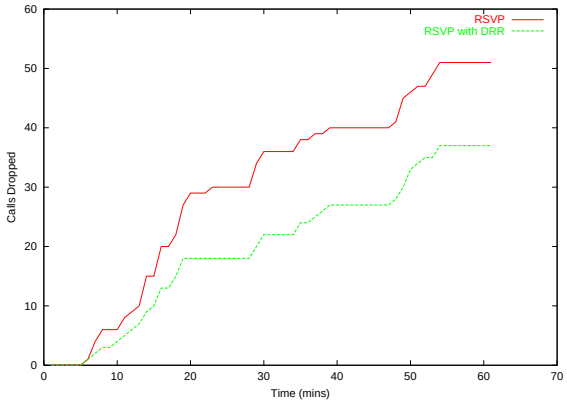
(a) QoS Request = 1 unit



(b) QoS Request = 2 units

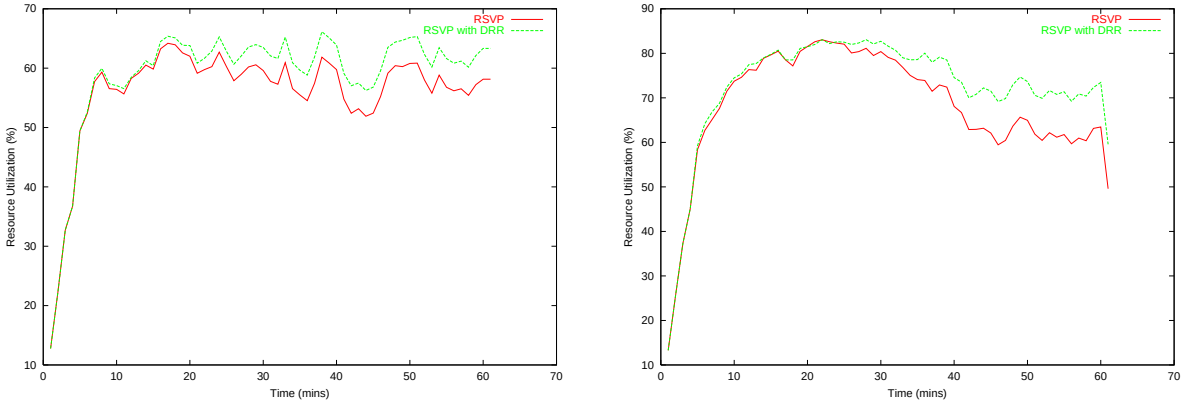


(c) QoS Request = 4 units



(d) QoS Request = 8 units

Figure 4.4: Drop Graph for Different QoS Requests for a Network Size of 30 Nodes with an Average Density of 5 (Non-Uniform QoS Request Distribution)



(a) Hold Time = 5 min

(b) Hold Time = 10 min

Figure 4.5: Resource Utilization for a Network Size of 30 Nodes with an Average Density of 5 (Non-Uniform QoS Request Distribution)

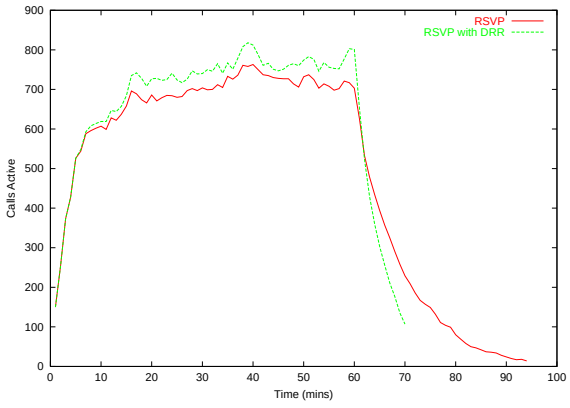


Figure 4.6: Time Complexity Experiment for a Network Size of 30 Nodes with an Average Density of 5 (Non-Uniform QoS Request Distribution)

Time complexity of an algorithm is the total time taken to execute the algorithm under given conditions. This experiment compares the time complexities of RSVP_DRR and RSVP. It is likely that a call will re-attempt to avail resources even after initially being denied. This holds true for both RSVP_DRR and conventional RSVP. This experiment projects the total time required to accommodate all calls in the network. The resource request for a denied call is repeated in the network until successful. The likelihood of a call succeeding after having failed in RSVP_DRR is higher than in RSVP as seen in the experiment. This is due to the fact that resources are attempted to be redistributed in RSVP_DRR upon successive attempts to grant resources to the denied call, which increases the chances that the call is successful sooner. The results verify that all calls are indeed, more likely to succeed with RSVP_DRR, within a shorter period of time than in RSVP. Thus, the total time taken to execute RSVP_DRR is than RSVP.

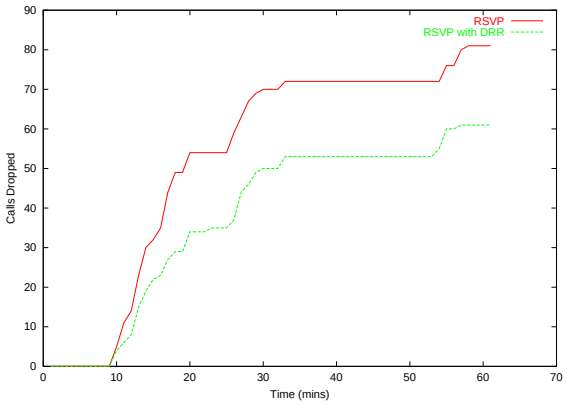
Figure 4.6 illustrates the time complexity, i.e., the time taken to accommodate all the call requests into the network. If a call request fails, it is attempted again after an exponentially distributed time interval with an average re-try time of 5 minutes. Initially, the same number of calls are active in the network for both RSVP and RSVP_DRR. This is because there are no call drops and all incoming calls can be accommodated as resources are available in the network. For subsequent calls, when

resources becomes scarce in the requested path, RSVP_DRR tries to redistribute resources to accommodate more calls. Thus, there is an increase in the number of calls active in the network for RSVP_DRR than the number of calls active in the network for RSVP. Consequently RSVP_DRR takes a shorter period of time to accommodate all calls as compared to RSVP, as it attempts to redistribute resources if necessary. RSVP does not redistribute resources, and therefore has to wait until resources are available in contiguous blocks. The last recorded time in the time complexity graph is the time at which all calls have been successfully granted resources by the network. RSVP_DRR takes approximately 70 minutes versus 94 minutes taken by RSVP to successfully accommodate all calls in the network (see Figure 4.6). Therefore, the redistribution of resources implemented in RSVP_DRR helps in reducing the total time taken to accommodate all calls in the network.

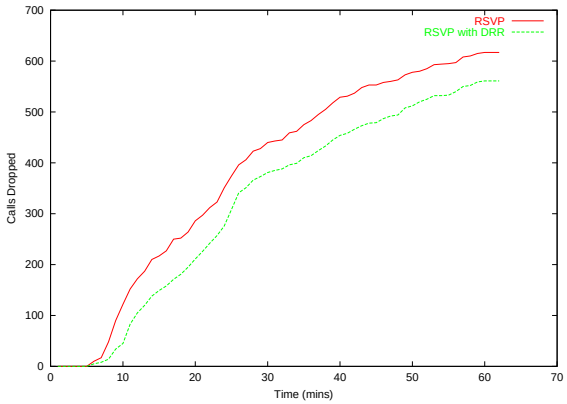
When the average density of the network is increased to 7, the number of calls dropped decreases in RSVP. This is due to the fact that there are more paths between any two nodes in the network. The RSVP_DRR algorithm also performs well as shown in Figure 4.7. The percentage difference in number of calls dropped between RSVP_DRR and RSVP is not as high as might be expected. This is due to the limitation imposed in the multiple next hop routing table that has only 4 possible next hop entries for a particular destination. In other words, maximally 4 alternate

paths are explored to redistribute resources. Since the average density of the network is 7, the number of alternate path choices has been limited, thereby curtailing the performance of RSVP_DRR. The results observed in Figure 4.8 (drops for different QoS requests), Figure 4.9 (resource utilization for different average call holding times) and Figure 4.10 (Time Complexity) mimic those that are observed in Figure 4.4, Figure 4.5 and Figure 4.6 respectively.

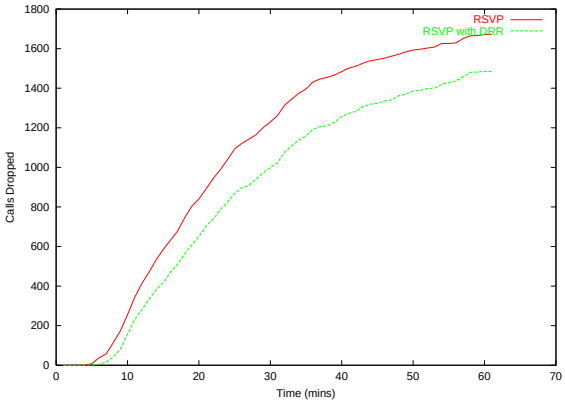
When the average density of the network is reduced to 3 (Figure 4.11), the total number of calls dropped using RSVP_DRR is more than that in RSVP. One plausible explanation is that in a sparse network, the number of nodes directly connected to other nodes is less. This means that there are few alternate paths that can be used for redistribution of resources. Even if a path is found to redistribute resources, another call originating at one of the nodes along the selected alternate path may have to be dropped because resources were redistributed on this path.



(a) Hold Time = 5 min

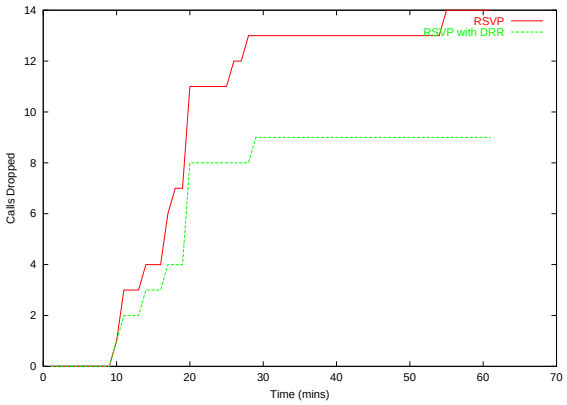


(b) Hold Time = 10 min

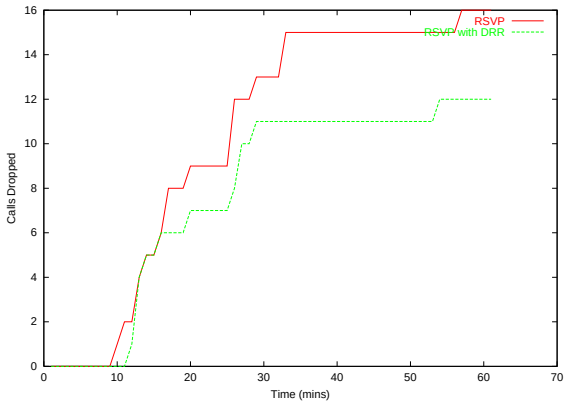


(c) Hold Time = 20 min

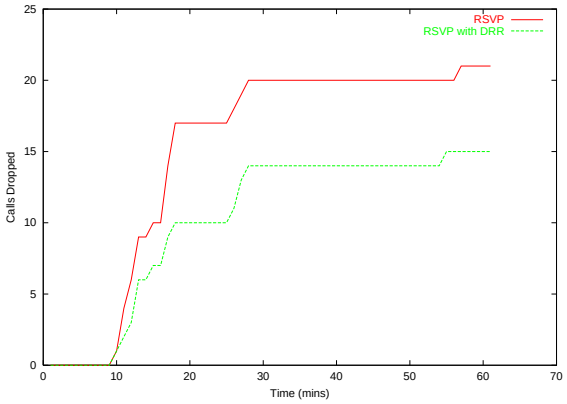
Figure 4.7: Drop Graph for a Network Size of 30 Nodes with an Average Density of 7 (Non-Uniform QoS Request Distribution)



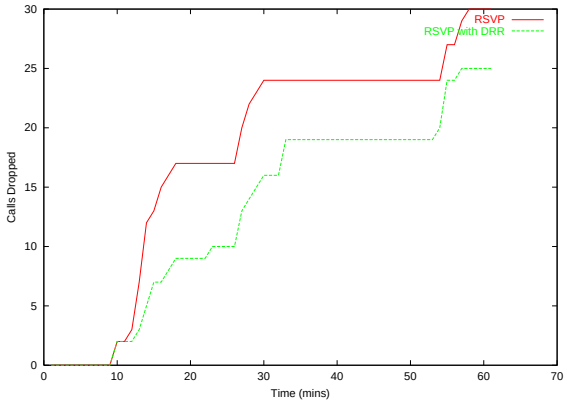
(a) QoS Request = 1 unit



(b) QoS Request = 2 units

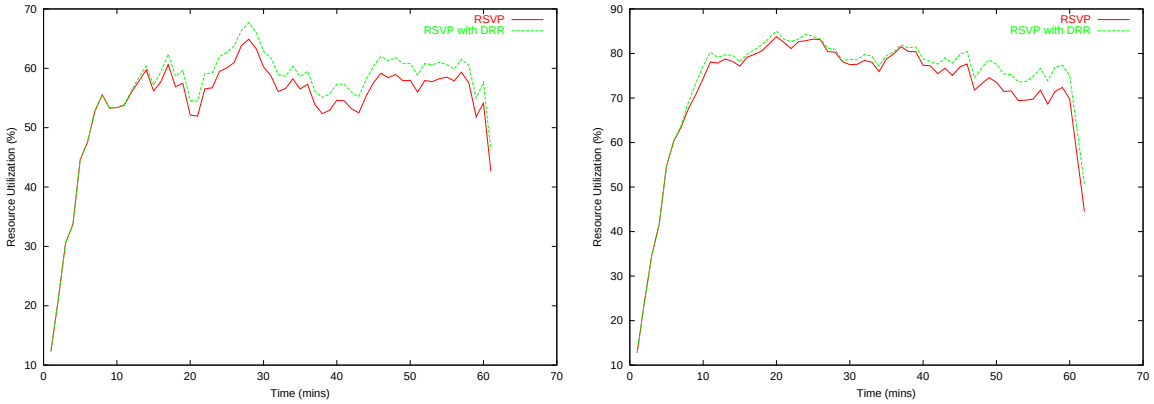


(c) QoS Request = 4 units



(d) QoS Request = 8 units

Figure 4.8: Drop Graph for Different QoS Requests for a Network Size of 30 Nodes with an Average Density of 7 (Non-Uniform QoS Request Distribution)



(a) Hold Time = 5 min

(b) Hold Time = 10 min

Figure 4.9: Resource Utilization for a Network Size of 30 Nodes with an Average Density of 7 (Non-Uniform QoS Request Distribution)

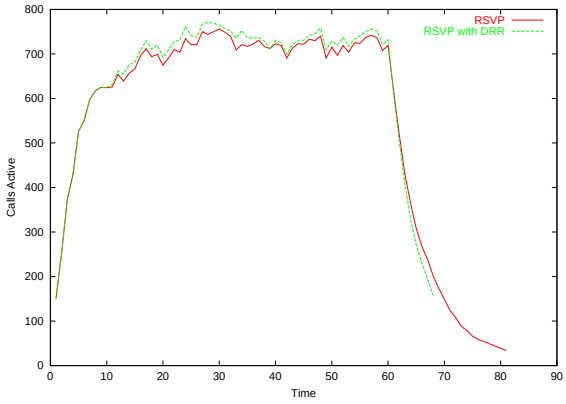
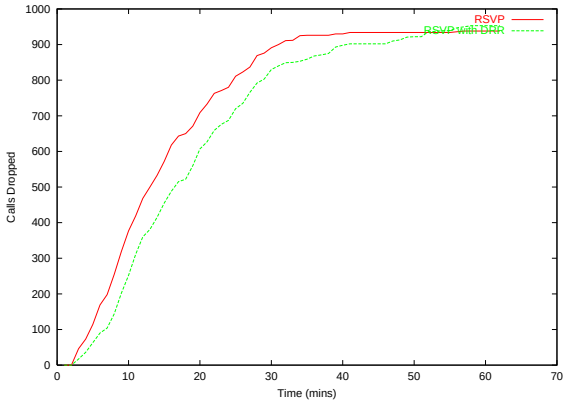
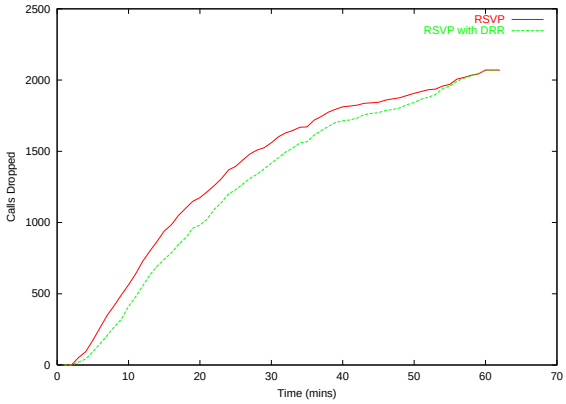


Figure 4.10: Time Complexity Experiment for a Network Size of 30 Nodes with an Average Density of 7 (Non-Uniform QoS Request Distribution)



(a) Hold Time = 10 min



(b) Hold Time = 20 min

Figure 4.11: Drop Graph for a Network Size of 30 Nodes with an Average Density of 3 (Non-Uniform QoS Request Distribution)

4.4 Uniform QoS Request Distribution

In this experiment, calls with QoS requests of 1, 2, 4 and 8 units of buffer space are generated with a probability of 25% each. The number of calls dropped in RSVP_DRR and RSVP were measured and compared, in order to evaluate the performance of RSVP_DRR. The results confirm that fragmented resources in the network increase the number of calls dropped in conventional RSVP, and that the redistribution of resources reduces the degree of fragmentation in the network. The experiments measure the performance of RSVP_DRR and RSVP in the scenario where the traffic is bursty with an equal probability as that of normal traffic.

The experiments reveal that there are more calls dropped here than in the non-uniform distribution of QoS requests. Lower class QoS requests are easier to accommodate in the network, than higher class QoS requests. In this experiment, the distribution of calls with a QoS request of 8 units is the same as that of calls with a QoS request of 1 unit. Hence, the number of calls dropped in this distribution increases.

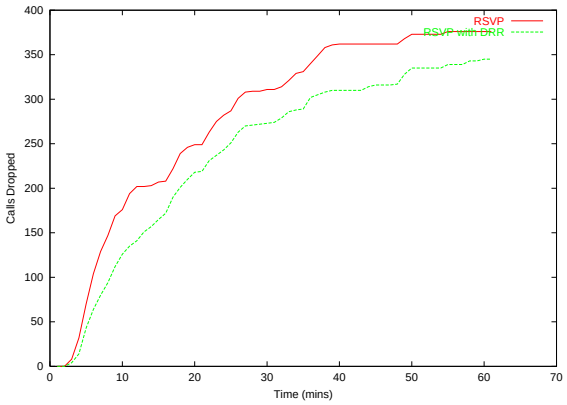
When the average density of the network is 5 (see Figure 4.12), RSVP_DRR compares favorably to RSVP in terms of number of calls dropped. However, the total number of calls dropped increases from that in the non-uniform QoS distribution.

This is due to the fact that more calls with resource requests of 8 and 4 units are generated and not all of them could be accommodated. The difference between the total calls dropped for RSVP_DRR and RSVP is less than that in the non-uniform QoS request distribution. Redistribution of resources of calls with a high QoS request is more difficult than that of calls with a low QoS request. Since calls with a high QoS request are generated with the same probability as calls with a low QoS, it is more difficult to accommodate the former in the network. This contributes to the decrease in the difference of total number of calls dropped between RSVP_DRR and RSVP. Irrespective of the call holding times, it is observed that the change in QoS request distributions affects the difference in number of calls dropped between RSVP_DRR and RSVP. The call holding times however, contribute to the increase in the total number of calls dropped i.e., we observe an increase in dropped calls for longer average call holding times.

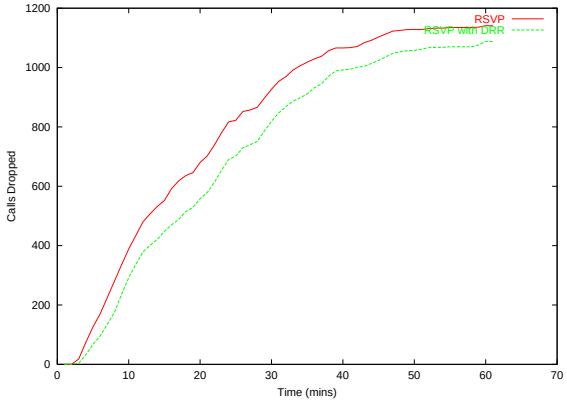
The total number of calls dropped in both RSVP_DRR and RSVP is higher than that in the non-uniform QoS request distribution, as explained in the earlier section. The difference in the number of calls dropped between RSVP_DRR and RSVP for QoS requests of 1 and 2 units (Figure 4.13) is almost the same as that for QoS requests of 1 and 2 units in the non-uniform QoS request distribution (Figure 4.4). Calls with a low QoS request can be easily accommodated, and resources for these calls are

easily redistributed. This explains the nature of this trend. However, it is observed that for QoS requests of 4 and 8 units, the number of calls dropped in RSVP_DRR increase, and is almost the same as that for conventional RSVP. In the non-uniform traffic distribution, the number of calls dropped in RSVP_DRR is less than that of RSVP (Figure 4.4). As explained earlier, resources for calls with a higher QoS are redistributed with greater difficulty than for those with a lower QoS request. Since there is an increase in the number of calls generated with a high QoS request in this traffic distribution, than in the non-uniform traffic distribution, they are more likely to be dropped. This increases the total number of calls dropped in RSVP_DRR. This indicates that calls with higher QoS requests contribute more to the total number of calls dropped in the network.

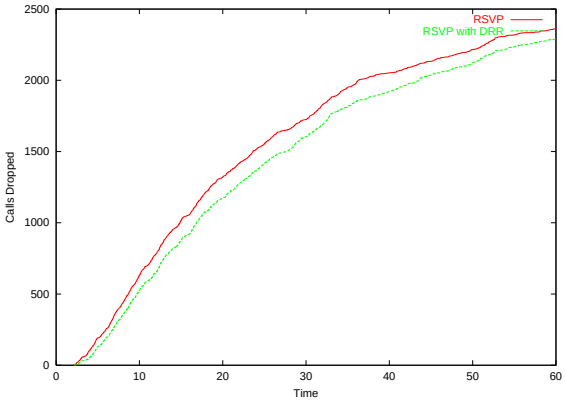
The experiment for time complexity in a uniform traffic distribution shows that the time taken to successfully accommodate all calls in the network is greater than the time taken in a non-uniform QoS request distribution (Figure 4.6), in both RSVP_DRR and RSVP. This difference is due to the fact there are more calls generated with QoS requests of 4 and 8 units, which are more difficult to accommodate in the network. The time taken to allocate and/or redistribute resources for these calls is longer, contributing to the increased time taken to accommodate all calls in the network.



(a) Hold Time = 5 min

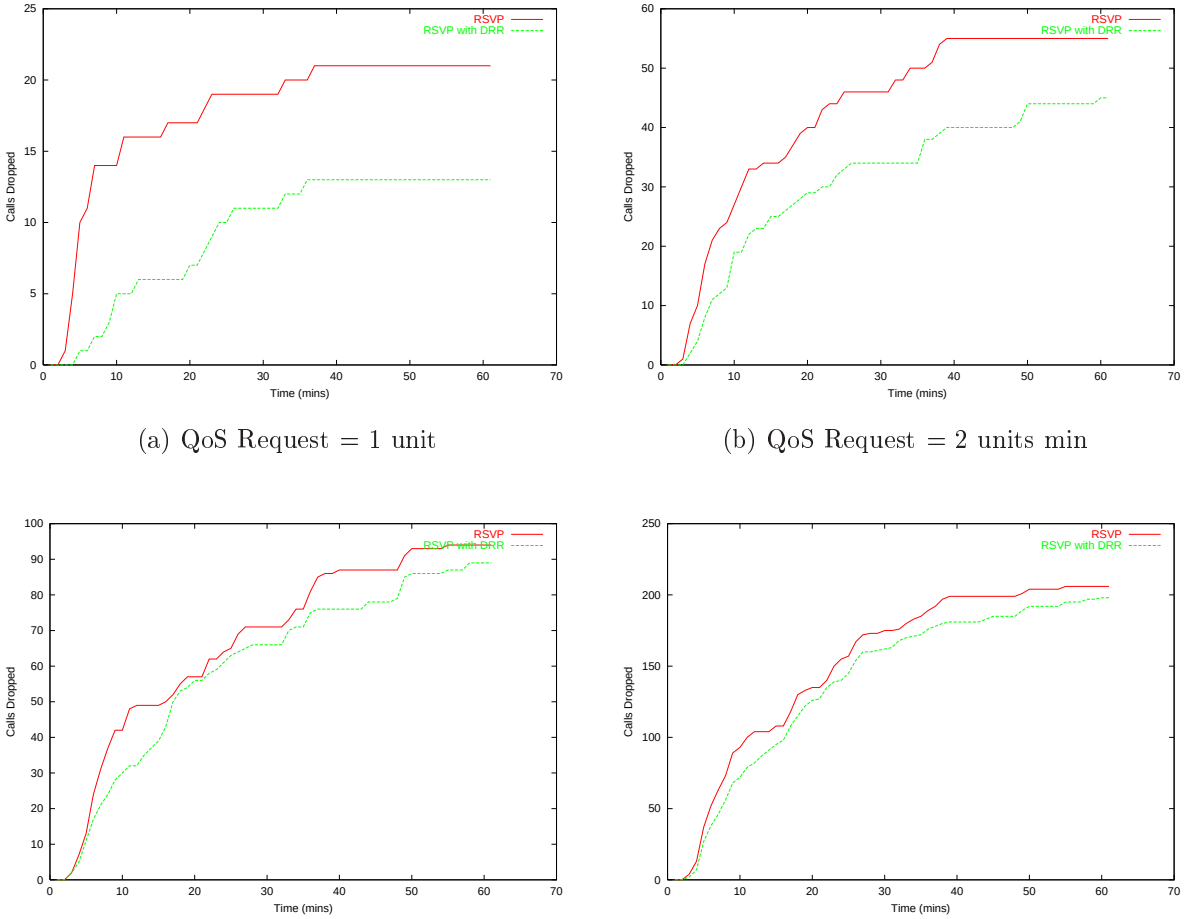


(b) Hold Time = 10 min



(c) Hold Time = 20 min

Figure 4.12: Drop Graph for a Network Size of 30 Nodes with an Average Density of 5 (Uniform QoS Request Distribution)



(a) QoS Request = 1 unit

(b) QoS Request = 2 units min

(c) QoS Request = 4 units min

(d) QoS Request = 8 units min

Figure 4.13: Drop Graph for Different QoS Requests for a Network Size of 30 Nodes with an Average Density of 5 (Uniform QoS Request Distribution)

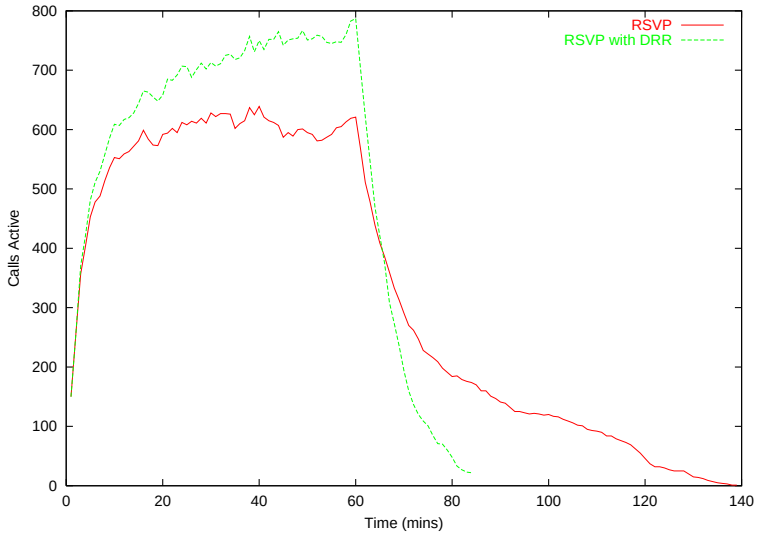


Figure 4.14: Time Complexity Experiment for a Network Size of 30 Nodes with an Average Density of 5 (Uniform QoS Request Distribution)

When the average density of the network is increased from 5 to 7, the total number of calls dropped decreases. However, other performance issues remain similar, and are comparative to the performance of RSVP_DRR and RSVP in a network of average density 5.

4.5 Message Overhead

Message overhead is a measure of the number of extra messages exchanged by nodes in the network due to the dynamic redistribution of resources. As seen in Figure 4.15, RSVP_DRR has a higher message overhead (44% more) than that of conventional

RSVP. The performance of RSVP_DRR, nonetheless offsets this increase in message count.

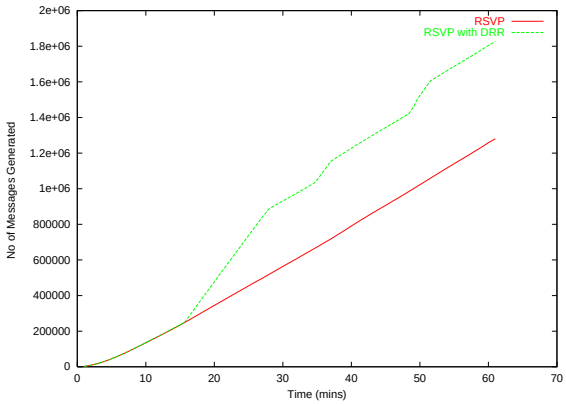


Figure 4.15: Message Complexity for a Network Size of 30 Nodes with an Average Density of 5

Implementing timers in the intermediate nodes may eliminate the exchange of some special messages thereby reducing the message overhead of RSVP_DRR. Optimizing the message overhead is left as future work.

4.6 Summary

RSVP_DRR, as seen in the experimental results, is more effective than conventional RSVP, as less calls are dropped. RSVP_DRR is especially effective in dense networks, where there are more paths available to be explored for redistribution of resources.

RSVP_DRR mitigates the degree of resource fragmentation, thereby contributing to increased performance.

CHAPTER 5

SUMMARY AND FUTURE WORK

5.1 Summary

Resource fragmentation leads to a significant number of dropped calls even when there are resources available in the network. Since resources are not available in contiguous blocks, not all incoming calls can utilize these resources. Resources may be available in small blocks, but cannot be allocated when they do not meet the total size requirements of incoming calls. The Dynamic Resource Redistribution algorithm implemented with RSVP (RSVP_DRR) reduces the effects of resource fragmentation in the network.

The results for both a non-uniform and uniform distribution of QoS requests indicate that RSVP_DRR is an improvement on RSVP in terms of the number of calls dropped. There are fewer calls dropped in RSVP_DRR than in RSVP, and resource utilization in the network increases. This is a direct consequence of reduced resource fragmentation when RSVP_DRR is used. The size of QoS requests of calls is also a factor in the performance of RSVP_DRR. When the QoS request is higher, calls are more likely to be dropped, as it is more difficult to accommodate such a

call, and redistribute resources for a large QoS request. It is also observed that the total time taken to accommodate all calls in the network is less in RSVP_DRR as compared to RSVP.

RSVP_DRR works well with high density [1]networks. The performance of RSVP_DRR also depends on the number of alternate paths available to a particular destination. Each next hop entry corresponding to a particular destination in the multiple next hop routing table characterizes a unique path to the destination. In other words, the number of next hop entries in the routing table of a node bounds the number of alternate paths explored for resource redistribution from that node.

The calls selected to be redistributed are chosen for their low QoS resource requests. The Flow Select algorithm selects such flows to be redistributed, that meet the resource requirement, either as a whole, or in multiples. The only consideration here is that few calls with low QoS requests can be redistributed more easily than many calls with high QoS requests between them. A single call or flow that satisfies the required QoS request is not selected to be redistributed in the very first attempt, as few flows with smaller QoS requests totalling the required QoS request are more likely to be redistributed in the network.

[1]RSVP_DRR is found to be more effective than RSVP when the average density of the network is 5 and 7 and RSVP is more favorable to RSVP_DRR when the average density of the network is 3

Thus the overall performance of RSVP_DRR in most cases is more effective than RSVP for unicast communications.

5.2 Future Work

RSVP_DRR, as explained in the previous sections, uses the Flow Select algorithm to select the flows whose resources may be redistributed in the network. In this specific algorithm, only an even number of flows is selected for resource redistribution. For example, when a flow with a resource request of 8 units of buffer space fails, the algorithm tries to select two flows with a resource allocation of 4 units each for resource redistribution.

Instead, it is also possible to redistribute one flow with a resource allocation of 4 units and 2 flows with resource allocations of 2 units each. Indeed, this may even be easier to redistribute, as the resource requests are small. On the flip side, however the algorithm is now trying to redistribute the resources of 3 flows as compared to 2 flows initially, thereby introducing more traffic in the network.

RSVP_DRR is implemented only for unicast communications in this thesis. However, RSVP itself was developed keeping multicast communications in mind, and it is currently being supported in many networks that facilitate multicast. Therefore RSVP_DRR needs to be extended to multicast communications in the future.

RSVP_DRR could be tested on other routing protocols to observe the effect of using some other metrics, such as resource-oriented metrics, in routing on resource redistribution.

The performance of RSVP_DRR is better than RSVP because resources are redistributed on alternate paths. The selection of the flows to be redistributed is dependent on the Flow Select algorithm. However, alternate paths to the destinations of these selected flows are selected at random. The selection of the alternate path is merely based on the destination node being one of the next hops in the multiple next hop routing table of the nodes. RSVP_DRR may perform even better if an intelligent route selection mechanism is used in future implementations.

Guaranteed Load Service (GLS) [13] of Integrated Service Architecture (IntServ) controls the queuing delay experienced by a datagram and guarantees that this delay does not exceed a maximum limit. This guarantee is valid only as long as the flow conforms to its traffic specification advertised before the data transmission. RSVP is an IntServ signaling protocol used to provide GLS. This factor (delay guarantee) is however not considered by RSVP_DRR while redistributing resources and exploring new paths for the flows of redistributed resources, as the number of intermediate nodes on the new path is unknown. The new path may be longer than the old path, thus possibly increasing the queuing delay guaranteed by GLS. The repercussions

of such an increase in queuing delay are yet to be investigated. This issue was not addressed in this work, since the experiments were conducted on the assumption that the maximum allowed queuing delay would not be exceeded. It is possible that in spite of being longer, the alternate path for the flows to be redistributed may not have a longer queuing delay. Nevertheless, it is a distinct possibility that this assumption may not be true in all cases, and consequently may adversely affect the guaranteed QoS.

The value of timer T1 used in the DRR algorithms needs to be tuned further. A timeout value of 1 second for T1 was used to make DRR scalable for large networks. This value could be fine tuned with the help of statistical delay information from the network.

The message complexity of RSVP_DRR is higher than that of RSVP. Some extra messages are used in RSVP_DRR for redistribution of resources, which contribute to the increased message complexity. Implementing timers to replace the explicit transmission of messages in some cases could reduce the increased message count in RSVP_DRR. For example, sending an explicit S_RTEAR message could be avoided, if timers were implemented in the nodes to release resources after a period of time, if the S_CONFIRM message were not received. This is an alternative to be explored in the future.

BIBLIOGRAPHY

[1] Aurrecoechea, C., Campbell, A., Hauw, L., "A Survey of QoS Architectures,"ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture, vol 6,pp. 135–151, May 1998.

[2] Braden, B., Ed., et. al., "Resource Reservation Protocol (RSVP) - Version 1 Functional Specification", RFC 2205, September 1997.

[3] Braden, R., Clark, D., Shenker, S., "Integrated Services in the Internet Architecture : an Overview", June 1994.

[4] Braden, R., Estrin, D., Berson, S., Herzog, S., and D. Zappala, "The Design of RSVP Protocol",ISI Final Technical Report, July 1996.

[5] Cisco Systems, "Introduction:Quality of Service Overview".

[6] Clark, D., " The Design Philosophy of the DARPA Internet Protocols", ACM SIGCOMM 88, August 1988.

[7] Ferguson, P., Huston, G., "Quality of Service in the Internet: Fact, Fiction, or Compromise ?" Proceedings of INET 98.

[8] Ferguson, P.,Huston, G., " Quality of Service - Delivering QoS on the internet and in Corporate Networks" ISBN 0-471-24358-2, 1998

[9] Floyd, S., Jacobson, V., "Random Early detection Gateways for Congestion Avoidance,", IEEE/ACM Transactions on Networking, v.1,n.4, August 1993

[10] Internet 2, "Internet Protocol Quality of Service Problem Statement", Internet Draft, September 1997.

[11] Microsoft Corporation, "Microsoft QoS Technical Review", September 13, 1999

[12] Moore,B., Ellesson, E., Strassne, J., Westerinen, A., "Policy Core Information Model - Version 1 Specification", February 2001.

[13] Shenker, S., Partridge, C., and R Guerin, "Specification of Guaranteed Quality of Service", RFC 2212, September 1997.

[14] Siegel, E., Passmore, D., "Network Quality of Service – What's Good Enough?"

[15] Stallings, W., "High Speed Networks - TCP/IP and ATM Design Principles", 0-13-525965-7, 1998.

[16] Stardust.com Inc, "White Paper - The Need for QoS", July 1999. September 06, 2001)"

[17] Stardust.com Inc, "White Paper - QoS protocols and Architecture",July 1999

[18] Rajan, R., et al.,"A Policy Framework for Integrated and Differentiated Services in the Internet",IEEE Vol 13 No.5, September/October 1999.

[19] The Quality of Service Forum(www.qosforum.com/docs/faq/)"The IP QoS FAQ", September 1999.

[20] Wroclawski, J., "Specification of the Controlled Load Quality of Service", RFC 2211, September 1997.

[21] Wroclawski, J., "The use of RSVP with IETF Integrated Services", RFC 2210, September 1997.

[22] Zhao, W., Olshefski, D., Schulzrinne, H., "Internet Quality of Service: an Overview", Columbia University, New York,Technical Report CUCS-003-00, Feb 2000.